GenRad
Development Systems Division

Z80 PERSONALITY MANUAL

2300-5013-01

MICROCOMPUTER
SYSTEMS

Z80 PERSONALITY MANUAL

2300-5013-01

June 1981

GenRad/Development Systems Division
5730 Buckingham Parkway
Culver City, CA 90230

REVISION HISTORY

Title	Number	Release Date	Notes
Z80 Personality Manual	2300-5013-00	2/81	First Edition
Z80 Personality Manual	2300-5013-01	6/81	Second Edition

RELATED PUBLICATIONS

ADS Software Reference Manual	2300-5000-01
Programming Reference Guide	2300-5001-01
2300 ADS Installation and Maintenance Manual	2300-5003-01
Z80 Self-Instruction Manual	2300-5025-00
Schematics Reference Manual	2300-5031-00

© GenRad/DSD, 1981

PREFACE

The Z80 Personality Manual is intended as a supplement to the 2300-ADS Software Reference Manual. Special operating procedures, installation information, and Z80 characteristics are described in this manual.

Please note that a **Documentation Reply Card** is included in this manual. When you complete and return it, you help us produce better documentation for you.

A **User Registration Card** is included in the set of manuals you receive with your GenRad/DSD system. When you complete and return the User Registration Card, you ensure that you will receive all updates and new information for your configuration.

For your convenience, a list of **GenRad/DSD Service Locations** is appended to this manual.

Digitized by the Internet Archive
in 2025

https://archive.org/details/manualsplus_06649

CONTENTS

	PAGE
CHAPTER 1 - DEBUGGER	
Introduction	1-1
Debugger Display	1-1
Register Display	1-3
Disassembly Display	1-3
Memory Map	1-4
Memory Pointers	1-4
Message Line	1-4
Command Line	1-4
Debugger Reconfiguration	1-5
Modifying the Command File	1-9
Interrupt Vector Modification	1-12
Symbolic Debugging	1-12
Breakpoint Facility	1-13
Example	1-13
Display During User Program Execution	1-14
CHAPTER 2 - EMULATION	
Introduction	2-1
Z80 Emulator Features	2-2
Sharing the Non-Maskable Interrupt	2-3
Example I	2-3
Example II	2-5
Sample Command File	2-7
Step-by-Step Debugging	2-8
Clock Test	2-8
Debugging Z80 Interrupts	2-8
System Operation	2-9
Verify User System Operation	2-10
Diagnostic Program Execution	2-11
User Program Execution (ROM/PROM)	2-11
User Program Execution (Target System)	2-12
System Operation with Microprocessor Installed	2-12
Considerations Prior to Use of Z80 In-Circuit Emulator	2-13
Control Lines and Their Effect	2-14
Installation	2-16
Z80 Probe - Phasing Adjustment Procedure	2-18

PAGE

CHAPTER 3 - UNIVERSAL LOGIC ANALYZER

Introduction 3-1
Logic Analyzer Entry and Exit 3-3
Logic Analyzer Display 3-3
Breakpoints 3-3
AND/OR Provision 3-3
Trace Qualifier 3-3
Timer Increment 3-4
Default Conditions 3-4
Example: Breakpoint and Trace Qualifier 3-7
Executing a Program with Logic Analysis 3-10
Executing from Logic Analyzer 3-10
Interpretation of Results 3-11
Parallel State Display 3-11
Serial Display 3-12
Logic Analyzer Installation 3-15
External Probe Assembly Connection 3-15
CPU and ICE Interface Card Connection 3-15
Strapping 3-18

CHAPTER 4 - EPROM PROGRAMMER

Introduction 4-1
Installation 4-4
Operation 4-4
Programming an EPROM 4-8
Command Parameters 4-8
Sample EPROM Programming Procedure 4-9

CHAPTER 5 - COMMANDS

Introduction 5-1
Formation of Expressions 5-1
List Of Commands 5-3
Load Program 5-3
Store Data in Memory 5-3
Display Memory 5-4
Display Positioning 5-4
Execute Program 5-5
No-Interrupt Execution 5-5
Find 5-6

PAGE

CHAPTER 5 - CONT'D

Single-Step 5-6
Split Display Mode 5-6
Trace Execution 5-7
Trace Recall 5-7
Write 5-9
Write-Protect Memory 5-9
Store Data in Register 5-10
Set Breakpoint 5-10
Display Breakpoint 5-11
Reset Breakpoint 5-11
Clear Breakpoints 5-11
Invoke Logic Analyzer 5-11
Invoke Debugger 5-11
Select AND/OR Mode 5-12
Set Timer 5-12
Set Breakpoint in Logic Analyzer Mode 5-13
Set Trace Qualifier 5-14
Select Display Format 5-15
Emulator Mode Command 5-16
User Memory Map Command 5-17
EPROM Programming Commands 5-18
Information/Error Messages 5-19

GenRad/DSD Service Locations

ILLUSTRATIONS

FIGURE		PAGE
1-1	Debugger Display	1-2
1-2	Linker Memory Map (Before Changes)	1-7
1-3	Linker Memory Map (After Changes)	1-8
2-1	Z80 In-Circuit Emulator Assembly	2-17
2-2	Adjust Potentiometer R4	2-19
2-3	Adjust Potentiometer R2	2-20
3-1	ADS Backview	3-2
3-2	Logic Analyzer Parameters Default Values	3-4
3-3	Parallel Display	3-13
3-4	Serial Display	3-14
3-5	Logic Analyzer Card Assembly	3-16
3-6	Logic Analyzer/External Probe Assembly	3-17
4-1	2704/2708 EPROM Programmer	4-2
4-2	2716/2532/2732 EPROM Programmer	4-3
4-3	Cable Insertion in the Debug Board	4-6
4-4	Debugger Memory Available in 48K System	4-7
4-5	Debugger Memory Available in 64K System	4-8
5-1	Sample Trace Execution Display	5-8

TABLES

TABLE		PAGE
1-A	Debugger Use of Interrupt Vectors	1-12
3-A	Breakpoint Parameters	3-5
3-B	Trace Qualifier Parameters	3-6
3-C	Strapping	3-18

Chapter 1

DEBUGGER

INTRODUCTION

The Debugger provides a powerful system for finding and correcting program errors. Features include:

a unique split-screen display which allows different areas of memory to be viewed simultaneously in both disassembled and memory map form;

the ability to change the Debugger's location in memory and to modify the Debugger's use of interrupt vectors;

the ability to single-step program execution from any point in the program, view and change the contents of memory and registers;

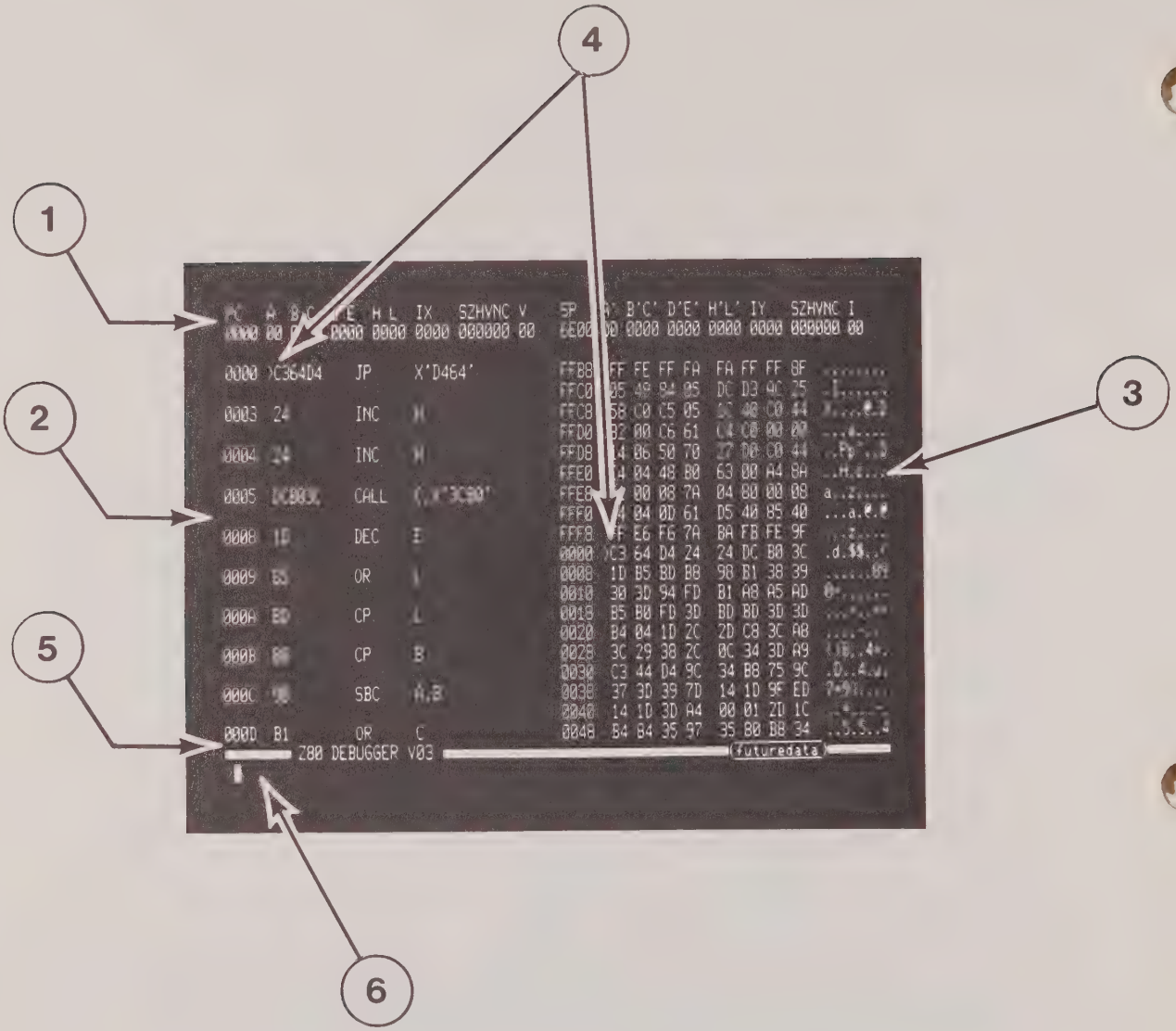
the use of breakpoints which may be set to stop execution and return control to the keyboard;

the ability to reference memory locations by the symbolic address labels contained in the user's source program;

a trace facility which tracks program execution and allows the user to view the state of the program counter, registers, and status bits for the twenty instructions executed prior to a breakpoint.

DEBUGGER DISPLAY

Figure 1-1 is a sample Debugger display showing a register display, a disassembly display, a memory map, two memory pointers, a message line and a command line.



1. Register Display
2. Disassembly Display
3. Memory Map
4. Memory Pointer
5. Message Line
6. Command Line

Figure 1-1. Debugger Display

REGISTER DISPLAY

The register display is shown in the top two lines of the Debugger display. The first line of the display contains the mnemonics of the internal registers for the Z80 microprocessor. They are:

Program Counter
Main Register Set
Index Register IX
Status Flags
Interrupt Vector
Stack Pointer
Alternate Register Set
Index Register IY
Alternate Status Flags
Instruction Register

The second line displays the contents of the registers (in hexadecimal) and program status flag bits.

The Stack Pointer register is initially set to a memory address just below the Debugger program. User programs may implement this Stack Pointer value when running under Debugger supervision. The user must set the Stack Pointer register to an available area of RAM before starting program execution.

DISASSEMBLY DISPLAY

Below the register display on the left side of the screen is the disassembly display consisting of ten microprocessor instructions in hexadecimal and the corresponding instruction mnemonics obtained by disassembly.

The disassembly display has four columns. The far left column contains the hexadecimal address of each instruction. If an instruction is longer than one byte, the address of the first byte is displayed. Write-protected address locations are appended with an asterisk (*). Locations mapped to areas in target system memory space during emulation are appended with a plus sign (+).

The next column displays instructions in hexadecimal. The third column shows the instructions in disassembled (mnemonic) form. The fourth column is the operand field, which shows register mnemonics and numeric values. Values assumed to be addresses are shown in hexadecimal, enclosed in single quotes and preceded by an X. All other values are displayed in decimal. The hexadecimal value can always be found in the second column.

If the portion of displayed memory contains data rather than an instruction, the disassembly display is meaningless. Therefore, to interpret correctly the information shown, the user must position the display at the first byte of an instruction.

MEMORY MAP

The right half of the Debugger display shows both an ASCII and a hexadecimal dump of 160 memory locations. The second through ninth columns display memory contents in hexadecimal. The first column represents the hexadecimal address of the byte displayed in the second column. The next sequential hexadecimal address (not displayed) is the location of the byte in the third column, etc. The far right side of the display contains the ASCII representation of the eight memory map locations. If there is no corresponding ASCII character, a dot is displayed.

MEMORY POINTER

The two memory pointers (one on each side of the display) point to the area of memory currently being viewed. When the Debugger is invoked, the contents of memory beginning at location X'0000' are displayed and both sides of the display are slaved together to show the same area of memory. The display can be split to view separate portions of memory simultaneously by depressing the TAB key and entering a command to display a new location.

MESSAGE LINE

The message line is the second line from the bottom of the display. Debugger messages, including error messages, appear here.

COMMAND LINE

The command line is the last line of the display. When the Debugger is invoked, one memory pointer and the blinking rectangular cursor are positioned at the start of the command line. Both sides of the display are slaved together, and any commands entered affect both sides. When the TAB key is depressed, the display is split and a second memory pointer appears in the middle of the command line, marking the beginning of the memory map. Subsequent depressions of the TAB key move the cursor to either display side. Commands entered in this mode only affect the side where the cursor is located. The RESET key followed by a Display command is used to return the Debugger to slaved display mode.

NOTE

The RESET key sets all registers except the Stack Pointer to zero and resets Emulator and Logic Analyzer breakpoints.

DEBUGGER RECONFIGURATION

The Debugger program is configured to load into the upper portion of memory in a 48K or 64K system. However, there are several circumstances which require relocation of the Debugger:

1) The target system uses RESTART6 (location X'30'). Software breakpoints must be moved from RESTART6 to another of the maskable interrupts (0-7).

2) The target system uses the Non-Maskable Interrupt (NMI) (location X'66'); therefore the NMI should be shared between the target system and the Debugger. **(See Chapter 3, IN-CIRCUIT EMULATION, Sharing the NMI).**

3) The user wants to use some portion of memory where the Debugger is located for ROM simulation memory.

4) The target system uses X'D400' to X'D4FF' (the default location of the Debugger); therefore the 256-byte communication block DEBG256 must be moved to another portion of memory.

All files needed to relocate the Debugger reside on the RDOS II Debugger/Logic Analyzer diskette (2300-6098-XX) and on the RDOS II Utility Files diskette (2300-6097-XX). The user will also need the RDOS II System Files diskette (2300-6906-XX), initially residing on drive 0. (The "XX" refers to software revision numbers, which may vary) The user should be thoroughly familiar with the memory and I/O map of the target system before attempting relocation. Please note the following restrictions:

No RSEGS may be loaded at the following locations:

A000 to BFFF (in a 48K system this area contains no memory)
E000 to FFFF (in a 48K system this area contains no memory)
(the above areas may be used in a 64K system)
E800 to F000 (this area may be used for ROM/PROM simulator memory if write-protected)
D500 to D5FF (contains the boot loader work area)
E000 to E7FF (contains the boot PROM; may be used for ROM/PROM simulator memory if write-protected))

The RSEGs listed below must be linked for loading at the locations specified:

RSEG WADATA (the command file processor work area) must be loaded at X'D600'. If it is not ORGed there, the ability to run command files that invoke the Debugger is lost.

RSEG DEBG256 (which cannot be mapped to user memory) should be loaded on a 256-byte boundary.

RSEG CRT must be loaded on a 2K-byte boundary.

There are two versions of the Debugger. One version contains the Debugger facility only; this version is relocated by the command file DBGZ80.LCF. The second version contains the Debugger, Logic Analyzer, and PROM programming capabilities; this version is relocated by the command file DBLZ80.LCF. Both of these command files are located on the Debugger/Logic Analyzer diskette.

When the new locations have been decided upon, the modifications should be entered into the appropriate command file. Load the file via the Editor with the L command. After modifications are made, write the command file back to diskette using the W command. See the section **Modifying the Command File**, this chapter, for further details.

Execute the command file to reconfigure the Debugger by entering the command:

```
J^1:DBLZ80.LCF
```

Print the Linker memory map and compare the new locations of the RSEGs with the list of restrictions to be sure there are no conflicts.

Two sample Linker memory maps are shown here. **Figure 1-2** shows memory locations prior to reconfiguration. **Figure 1-3** shows memory locations after the command file has been modified. Note that the Debugger program has been moved to location X'200' in the lower portion of memory, LALOAD has been ORGed at X'32DA', and DEBG256 has been relocated to X'4700', leaving X'4800' to X'9FFF' and X'C000' to X'D5FF' (in a 48K system) for the user's emulation memory.

ADDR	RSEG	FILE	LENGTH
6E00	DEBUGR	DEBUGZ80.R	158D
838D	PROM	PROMZ80.R	0226
85B3	BRKRST	BRKRSZ80.R	000C
85BF	ANALYZ	ANALYZ80.R	05F3
8BB2	SETPARM	LASCAN80.R	037E
8F30	LADCON2	LADCN380.R	0091
8FC1	LAPAR	LAPARD80.R	0323
92E4	LASERD	LASERD80.R	0174
9458	RDLAD	RDLAD80.R	00A6
94FE	DSKCODE	DISK80.R	09DC
C000	LALOAD	LALOAD80.R	030E
C30E	KEYIN	KEYIN80.R	0294
C5A2	LINEIN	LINEIN80.R	00EE
C690	KIO	KIO80.R	01D2
C862	IOCODE	DEXEC80.R	0004
C866	DISAM80	DISMZ80.R	0352
CBB8	TABLE1	TABZ80.R	07C8
D400	DEBG256	DEBUGZ80.R	00EB
D600	WADATA	WORK.R	0200
D800	CRT	KIO80.R	07D0
D800	DCRT	DEBUGZ80.R	07B0

Figure 1-2. Linker Memory Map
(Before Changes)

ADDR	RSEG	FILE	LENGTH
0200	DEBUGR	DEBUGZ80.R	158D
178D	PROM	PROMZ80.R	0226
19B3	BRKRST	BRKRSZ80.R	000C
19BF	ANALYZ	ANALYZ80.R	05F3
1FB2	SETPARM	LASCAN80.R	037E
2330	LADCON2	LADCN380.R	0091
23C1	LAPAR	LAPARD80.R	0323
26E4	LASERD	LASERD80.R	0174
2858	RDLAD	RDLAD80.R	00A6
28FE	DSKCODE	DISK80.R	09DC
32DA	LALOAD	LALOAD80.R	030E
35E8	KEYIN	KEYIN80.R	0294
387C	LINEIN	LINEIN80.R	00EE
396A	KIO	KIO80.R	01D2
3B3C	IOCODE	DEXEC80.R	0004
3B40	DISAM80	DISMZ80.R	0352
3E92	TABLE1	TABZ80.R	07C8
4700	DEBG256	DEBUGZ80.R	00EB
D600	WADATA	WORK.R	0200
D800	CRT	KIO80.R	07D0
D800	DCRT	DEBUGZ80.R	07B0

Figure 1-3. Linker Memory Map
(After Changes)

MODIFYING THE COMMAND FILE

The command file DBLZ80.LCF containing the Logic Analyzer, Debugger, and PROM programming capabilities is supplied on the RDOS II Debugger/Logic Analyzer diskette. To modify this file, load it to the Editor with the command:

```
L1:DBLZ80.LCF
```

The command file sequence is as follows:

```
JE                          The command file jumps to the Editor, loads
L1:BRKRSZ80.N               the source file BRKRSZ80.N and searches for
F/******/                   the software breakpoint parameter.  The user
DEL                         may change the parameter at this point.
I
RSTNUM    EQU 6             ******SOFTWARE BREAKPOINT
                                  MAY BE CHANGED (0-7)

W1:BRKRSZ80.N               Write the source file back to diskette.
JA                          The Assembler is invoked and the options
ET                          E(list errors only) and T(truncated list) are
                            entered

1:BRKRSZ80.N                input file (source file)

1:BRKRSZ80.R                output file for Assembler
```

When the Assembler completes, the Manager reattributes the Debugger.

```
JM
A1:DEBUGZ80L/A=0
```

The Linker is invoked next and an escape character (^) is entered to halt command file processing. This allows the user to specify additional Linker options if desired and also to put the RDOS II Utility Files diskette into drive 0. (This is necessary only when actually executing the command file.)

```
JL
L^L
```

The command file enters the following as input files for the Linker:

```
1:DEBUGZ80.R                main Debugger program; includes DEBG256,
                            command selector, software routines, symbol
                            table routines, initialization code

1:BRKRSZ80.R                software breakpoint selection routine

1:PROMZ80.R                 PROM programming routines for all PROM
                            programmers
```

1:ANALYZ80.R	Logic Analyzer initialization, command scanner, breakpoint parameter evaluation, breakpoint display generation,and command routines
1:LASCAN80.R	Process breakpoint parameters and build control block
1:LAPARD80.R	Logic Analyzer parallel (disassembly and state) display generator
1:LADCN380.R	Convert 24-bit binary to decimal
1:LASERD80.R	Logic Analyzer serial (waveform) display generator
1:RDLAD80.R	Read in Logic Analyzer trace memory
1:LALOAD80.R	Load the Logic Analyzer match counters, timers, and match memory from control block generated by LASCAN
1:DISMZ80.R	Disassembler processor
1:TABZ80.R	Disassembler tables
DEXEC80.R	Dummy I/O executive for Debugger and Logic Analyzer
KEYIN80.R	Process keyboard input from both keyboard and command file
LINEIN80.R	Process an input command line
KIO80.R	CRT display driver
DISK80.R	Disk I/O driver
WORK.R	Command file processor work area
<blank line>	No more input files
1:DEBUGZ80L	The output file
^L	The command file pauses to allow user to enter a listing file or direct the listing to the printer.

The following are locating commands to put the Debugger as far up in memory (in a 48K system) as possible. If users wish to change any of these locations, they may do so here (except for those noted as restricted):

```
#ORG X'6E00'        48K system - no memory at X'A000' - X'BFFF'
DEBUGR, PROM, BRKRST, ANALYZ, SETPARM, LADCON2, LAPAR, LASERD, RDLAD,
DSKCODE
```

The following ORG command is necessary to skip over X'A000' to X'BFFF':

```
#ORG X'C000'
LALOAD, KEYIN, LINEIN, KIO, IOCODE, DISAM80, TABLE1
```

DEBG256 must be on an even-byte boundary:

```
#ORG X'D400'        256-byte area which cannot be mapped
DEBG256
```

WADATA cannot be moved without losing command file processing:

```
#ORG X'D600'        512-byte work area must be at X'D600'
WADATA
```

CRT must be on an even 2K boundary:

```
#ORG X'D800'        2K CRT area must be on 2K boundary
CRT
#ORG X'D800'        CRT area defined in DEBUGZ80.R
DCRT
#END
```

The Manager is then called to reattribute the Debugger:

```
JM^L
A1:DEBUGZ80L/A=Z
```

After all changes are made, users should execute the command file to reconfigure the Debugger. Enter the command:

```
J^1:DBLZ80.LCF
```

Initially, the RDOS II System Files diskette should be in drive 0 and the RDOS II Debugger/Logic Analyzer diskette should be in drive 1. When command file processing pauses, remove the System Files diskette from drive 0 and insert the RDOS II Utility Files diskette in that drive. When the command file processing pauses a second time, remove the Utility Files diskette from drive 0 and insert the System Files diskette in that drive.

INTERRUPT VECTOR MODIFICATION

Three interrupt vectors are utilized by the Debugger features as listed in Table 1-A.

TABLE 1-A. DEBUGGER USE OF INTERRUPT VECTORS

Function	Vector Location
RESET key	RST (X'00')
Hardware Interrupts (includes Logic Analyzer Interrupts, BREAK key, Hardware Breakpoint, Trace Function and STEP key)	NMI (X'66')
Software Breakpoint	RESTART6 (X'30')

The software breakpoint vector may be changed according to user needs, to any one of the software restarts (0-7), by modifying the appropriate command file (as previously described) and relinking the Debugger via command file execution.

NOTE

If RST 0 is used, Debugger use of the RESET key will be lost.

SYMBOLIC DEBUGGING

Symbolic debugging allows memory locations to be referenced by their symbolic names rather than by absolute addresses. If a source program has been edited since the last listing was made, and absolute addresses are unknown, symbolic referencing can save a considerable amount of searching or listing time.

Symbolic debugging uses the symbol table generated during Compiler, Assembler, and Linker operations. The table relates program symbol names with their memory addresses. It may be loaded at a specified location with the L command:

L{filename}[,offset][,symbol table address[,G]]

When the symbol table has been loaded, the Debugger displays a message indicating table length in hexadecimal. The symbol table may be displayed using the D command to position the cursor on the right side of the screen at the symbol table address stated in the Load command. Each symbol table entry contains the one-to-eight-byte ASCII symbol, one byte of flags, and a two-byte address.

The first symbol table entry for each file is preceded by eleven bytes (one byte of flags followed by the ten-character filename). Load the symbol table into RAM area that is not used by the program or by the Debugger.

BREAKPOINT FACILITY

A breakpoint instruction stops program execution and returns control to the Debugger. Breakpoint instructions are particularly helpful for debugging lengthy programs since they can be used to divide the programs into sections which may then be executed separately. Problem areas may be localized and errors quickly located and corrected.

The Set Breakpoint command allows one hardware, and up to four software breakpoints to be specified. A breakpoint must be set at the first byte of an instruction. The format is as follows:

BS{n}

where n is an integer 0-4 inclusive.

A hardware breakpoint instruction (n=0) does not alter the program in any way. A software breakpoint instruction (n=1-4) temporarily replaces the first byte of the program instruction at the specified breakpoint location. This program instruction byte is stored in the breakpoint table until the breakpoint is reset, using the BR (Reset Breakpoint) command, or the table is cleared, using the BC (Clear Breakpoint) command. Instruction replacement does not affect program execution since the replaced instruction is not in the portion of the program being debugged. By resetting the breakpoint, the replaced instruction byte is returned to the program and execution may continue from there. If the BC command is used, the instruction is also cleared from the table and is not replaced in the program. For further detail on breakpoint commands, refer to **Chapter 5, COMMANDS.**

NOTE

When a breakpoint returns control to the Debugger, the displayed memory pointer may not necessarily be positioned at the breakpoint. To resume execution at the breakpoint location, perform a Breakpoint Display and a Breakpoint Reset (BD and BR).

EXAMPLE

A breakpoint is set at the first byte of a subroutine call instruction. When the breakpoint is reached, the subroutine call is not executed; the program is halted and control is passed to the Debugger. The parameters to be passed to the subroutine may now be viewed and changed if necessary.

To continue debugging, another breakpoint may be set at the instruction following the subroutine call, the instruction byte of the first breakpoint may be restored using the BR command, and execution resumed. The entire program may be sequentially debugged with this "leapfrog" method.

DISPLAY DURING USER PROGRAM EXECUTION

Before beginning program execution, be sure that actual instructions and not random data are to be executed. The Debugger blanks the display screen at the start of execution. The blank screen is usually not noticeable if execution requires a short period of time. The Debugger display returns when execution terminates (via a breakpoint or depression of the BREAK key).

The Trace command may be used or the program may be single-stepped if it is necessary to view the register contents during execution.

If the display remains blank for an unreasonably long period of time, press the BREAK key to return control to the Debugger. Memory and register contents will be frozen at the time the BREAK key is depressed. If the BREAK key is ineffectual, press RESET. The RESET key should return the slaved Debugger display to the screen and set all registers except the Stack Pointer to zero. If both BREAK and RESET fail to restore the normal Debugger display, press the LOAD key. LOAD will return the ADS to its initial power-up condition and will restore control to the user. To resume Debugger operation, the Debugger and the user program must be reloaded.

NOTE

The LOAD key should be used if memory contents outside the user area have been altered.

Chapter 2

IN-CIRCUIT EMULATION

INTRODUCTION

The In-Circuit Emulator is a hardware feature enabling the microprocessor in the ADS to emulate the microprocessor in the user system. A connector on the end of the Emulator cable is plugged directly into the microprocessor socket on the user prototype.

The purpose of the Emulator is threefold. It can be used to substitute RAM for the PROM/ROM in the user system. The RAM can be much more easily modified than the ROM/PROM, thus facilitating changes in user programs during development phase. The Emulator also provides debugging facilities in the user environment. Features such as single-step, trace and breakpoints are all included in the Emulator package. This package allows the user to gradually bring up the target system. The clocks, DMA facilities, interrupts and I/O and memory mapping can all be separately enabled.

When combined with the Universal Logic Analyzer, the In-Circuit Emulator/Analyzer combination provides three complex breakpoint registers with up to 24 address, 16 data, and 8 control lines. The Analyzer also allows real-time tracing of 256 bus transactions.

The Emulator package consists of the following elements:

1. An In-Circuit Emulator interface card (Part No. 2300-4712).
2. Cable(s) to connect the interface card to the processor card (Part No. 2300-0303 without Universal Logic Analyzer, Part No. 2300-0322 with Universal Logic Analyzer).
3. An In-Circuit Emulator Probe assembly (phase-adjustable, Part No. 2300-4018).

Refer to **Installation Section**, this chapter, for further information.

Initially, the ADS is used to debug and develop user software. The Emulator is then used to verify the proper operation of user hardware. The designer may switch clocks, control lines and memory from the ADS to the user (target) system. As each element of the user system is verified, it can be utilized until, finally, the Emulator connector is removed. This step-by-step procedure is extremely effective, since only one element at a time is being tested.

All of the ADS Debugger commands are available to the user during emulator operation. Thus, the user is able to examine and alter memory and registers, start a program, set breakpoints, single-step, trace, load and dump object programs.

Z80 EMULATOR FEATURES

The following is a summary of features of the Z80 In-Circuit Emulator:

1. Real-time emulation and debugging at 4Mhz clock cycles using high-speed static RAM.

2. Separate control over clock source, DMA, interrupts and I/O, memory map and refresh enable.

3. Memory mapping in 256-byte blocks under software control.

4. Hardware breakpoint, four software breakpoints, single step, trace execution functions.

5. When the Logic Analyzer is added, the combination provides three complex breakpoints consisting of:

   16 address lines
   8 data lines
   Memory - I/O
   R/W
   Instruction fetch (M1 cycle)
   DMA cycle
   four conditions dependent on states of external probes
   trace qualifier
   256 deep trace memory

6. A system with 48K bytes of RAM provides 28K bytes of ROM/PROM simulator memory from locations X'000' to X'6FFF'. A system with 64K bytes of RAM provides 44K bytes of ROM/PROM simulator memory from locations X'000' to X'8FFF'and X'E000' to X'FFFF'.

7. The host system and target system can share the Non-Maskable Interrupt (NMI) at location X'66'.

SHARING THE NON-MASKABLE INTERRUPT

The 2300 ADS uses the Non-Maskable Interrupt (NMI) line for system functions. Sharing the NMI line with the target system is accomplished in several different ways, depending on the application. Several examples follow:

EXAMPLE I

The target system has PROM at zero, uses the NMI, and at location X'66' there is a jump to the NMI interrupt routine.

Example:

Contents of PROM Address:

Address	Contents	Disassembly	
64	00	NOP	
65	00	NOP	
66	C3	JP X'1037'	; user program interrupt routine
67	37		
68	10		
69	00	NOP	
6A	00	NOP	

Sequence of Commands to Share the NMI:

Command	Explanation
JD	Jump to the Debugger.
D66	Display location at X'66'. Note the address of the jump instruction. This address is DEBG256 +X'0B' for the Z80 Debugger.
U0,01	Set the memory map mask so that the first 256 bytes from X'00' to X'FF' will be mapped into the target system.
MCM	Enable the memory map and enable the target clock. This allows examination of the first 256 bytes of target memory. The clock is mapped in case the emulation clock and the target clock run at different speeds.
WN:PROM,0,FF	Copy the first 256 bytes of memory into a new disk file called PROM.
M	Clear the emulation modes so that all resources are located in the development system.
U0,0	Clear the memory map mask (optional).
LPROM	Load the file PROM. Now addresses X'0' to X'FF' in 2300 RAM are a copy of addresses X'0' to X'FF' in the target system.
D66	Display location X'66'.

Note the address of the target NMI routine (in this case, X'1037').

SC3,ll,hh	Store a jump to address X'D40B' or DEBG256 + X'0B', where "ll" refers to the low-order byte and "hh" refers to the high-order byte of the destination address.

Next, store the address of the user's NMI routine at DEBG256. The address of the start of DEBG256 is found by subtracting X'0B' from the address stored at location X'66'.

Dhh,ll	Display the start of DEBG256.
S37,10	Store the address of the target system's NMI routine, low-order byte first.
WN:PROM.MOD,0,FF,D400,D401	Save the modified PROM and Debugger in a new disk file called PROM.MOD

EXAMPLE II

The target system has PROM at zero and uses the NMI. The NMI interrupt routine begins at location X'66'. The general solution is a cut-and-patch procedure.

Relocate the first three instructions of the beginning of the interrupt routine to an unused portion of memory. Store the jump to DEBG256 + X'0B' at X'66' and store the relocated start address of the interrupt routine, formerly at X'66', at DEBG256.

Example:

Contents of PROM Address:

Address	Contents	Disassembly
63		
64		
65		
66	F5	PUSH AF
67	DB	IN A,(X'6')
68	06	
69	E6	AND X'1'
6A	01	
6B	C2	JP NZ,X'346'
6C	46	
6D	03	
6E		
6F		
70		

Assume that the target system does not use memory above X'6000' (i.e., all memory is located between X'0000' and X'5FFF').

EXAMPLE II (continued)

Sequence of Commands to Share the NMI:

Command	Explanation
JD	Jump to the Debugger
D66	Display location X'66'. Note the address of the jump instruction. The address is DEBG256 + X'0B' for this Debugger.
U0,01	Set the memory mask so that the first 256 bytes of memory will be mapped into the target system.
MCM	Enable the memory map and the target clock. This allows examination of the first 256 bytes of the target memory space. The target clock will be mapped in case it is different from the 2300 clock.
WN:PROM,0,FF	Save the first 256 bytes of memory in a new disk file called PROM.
M	Clear the emulation modes.
U0,0	Clear the memory map mask
LPROM	Load the copy of the user PROM from diskette into 2300 RAM.
D66	Display location X'66'.
SC3,ll,hh,	Store a jump to DEBG256 + X'0B', where "ll" refers to the low-order byte and "hh" refers to the high-order byte of the jump instruction.
D6000	Display X'6000' which will be the start of unused memory.
SF5,DB,06,C3,69,00	Store copy of code which started at X'66' and a jump past the NMI interrupt vector.
Dhh,ll	Display the start of DEBG256. The address of the start of DEBG256 is found by subtracting X'0B' from the address in the jump instruction stored at X'66'.
S00,60	Store new start address of the target system's interrupt routine.
WN:PROM.MOD,0,FF,6000, 6005,D400,D401	Save the patches in a new file called PROM.MOD.

SAMPLE COMMAND FILE

This example displays how a command file is written to map memory for emulation. No comments are allowed in a command file. For further details and an example regarding the User Memory Map Command, please refer to **Chapter 5, COMMANDS.**

```
JD                      Jump to the Debugger
LPROM.MOD               Load patches to the target system
U0,FE                   Map all memory except modified first 256 bytes.
U800,FF
U1000,FF
U1800,FF
U2000,FF
U2800,FF
U3000,FF
U3800,FF
U4000,FF
U4800,FF
U5000,FF
U5800,FF
U6000,FE                6000 - 6100 for scratch area
U6800,FF
U7000,FF
U7800,FF
U8000,FF
U8800,FF
U9000,FF
U9800,FF
UA000,FF
UA800,FF
UB000,FF
UB800,FF
UC000,FF
UC800,FF
UD000,FE                DEBG256 area
UD800,FF
UE000,FF
UE800,FF
UF000,FF
UF800,FF
X                       Write-protect 0-FF
MER
```

When an NMI occurs, the Debugger checks the cause of the interrupt (Logic Analyzer breakpoint, hardware breakpoint, or BREAK key). If the interrupt is due to none of these causes, the Debugger will jump to the user's interrupt routine via the address stored at DEBG256. This applies to both examples previously discussed.

STEP-BY-STEP DEBUGGING

To develop hardware and software expeditiously, designers should validate the hardware and software at each step in the design process. Designers should keep in mind the basic goal of separating hardware and software problems. Thus, they should attempt to verify as much as possible of the hardware system by means of simple tests using unambiguous debugging programs. If this goal is kept in mind, it should be possible to verify most of the hardware system prior to executing the actual system program.

There are seven distinct steps in the design verification of a microprocessor-based system. To verify design, users should:

1. Verify operation of the ADS using clocks generated by the user system.

2. Verify operation of the microprocessor when the user control lines are active (DMA request, Interrupt request(s), Wait, Reset, Ready, etc.).

3. Verify basic operation of the system, using very simple programs.

4. Verify user I/O and memory using diagnostic programs resident in the ADS ROM/PROM simulator memory.

5. Execute the user program with all of the memory space in the ADS and all of the I/O space in the user system.

6. Execute the user program with all the memory space and all the I/O space mapped into the user system.

7. Execute the user program with all memory, I/O, and the microprocessor installed in the user system.

CLOCK TEST

The clock rise and fall times and levels should meet the requirements of the microprocessor manufacturer. An oscilloscope may be used to compare the clock measurements to that of the chip manufacturer. It should then be possible to plug in the Emulator probe and switch the clock line input(s) to the user system. If the clock signal from the target system does not appear to the ADS to be the same as the system clock, the ADS will halt operation.

If the system will not run using the user clocks, check (1) clock signals with the Emulator probe plugged in, or (2) the possibility of ground loops.

DEBUGGING Z80 INTERRUPTS

The Z80 CPU supports three types of interrupts. Mode 0 is equivalent to the 8080A RST instruction. The interrupt vector is interpreted as a single-byte object code, representing the first instruction to be executed following the interrupt acknowledge.

In Mode 1, the Z80 automatically assumes the next instruction executed following an interrupt response will be a Restart, branching to X'38'. In Mode 1, no interrupt vector is required.

In Mode 2, an eight-bit interrupt response vector from the interrupting controller is combined with the contents of the V (interrupt vector) register to form a 16-bit address. The contents of the V register comprise the high-order eight-bits and the interrupt response vector the low-order eight-bits. The contents of that address and of address + 1 are loaded into the program counter, forming the address of the first opcode fetch (start of interrupt routine).

The Mode 2 interrupt is unique to the Z80, and is used in conjunction with the RETI (return from interrupt) instruction. An interrupting controller such as the Z80 serial I/O controller will keep its interrupt request logic active until the RETI instruction has been executed. At this time, the controller sees an X'ED' followed by a X'4D' on the data bus. This alerts the controller that the interrupt has been serviced and the interrupt logic can be disabled. The sequence is similar if more than one controller is daisy-chained and interrupts are prioritized.

To perform Mode 2 interrupts, both the user clock and I/O (and interrupts) must be enabled, regardless of whether the program is executing out of host or target memory. If the program is executing from target memory, memory must also be mapped. The RETI instruction must be in target memory.

SYSTEM OPERATION

Microprocessor input control lines fall into three categories:

1. Interrupt and Reset inputs
2. DMA (Direct Memory Access) requests
3. Asynchronous wait inputs

The Emulator Debuggers allow the user to separate control of the DMA requests from the Interrupt, Reset and Wait inputs, since debugging of DMA operation differs from Interrupt and Reset functions. The user must refer to the parameters as described in the M (Mode) command to determine the proper command to verify operation with the control inputs. When verifying the control inputs, users should be aware of:

1. The use of the Reset input by the Debugger to clear the microprocessor registers. This is modifiable by users to cause their reset sequence; however, users must remember to modify the reset vector or instruction to cause their sequence.

2. The use of one or more of the microprocessor interrupt vectors, pins etc. by the Debugger for hardware breakpoint, software breakpoint, trace, single-step and BREAK key. Users may modify this action, but must be aware of the debugging features that may be lost. For example, if the interrupt vector for the software breakpoint is changed from X'30' (RESTART 6) to X'0' (RESTART 0), the use of the RESET key will be lost.

3. The user system cannot have an interrrupt pending at the onset of user program execution, or that interrupt will be acknowledged immediately before any code is executed. This includes any code to mask interrupts, set interrupt vectors, set interrupt priorities, etc. The "N" (NO-INTERRUPT) command is very useful in this instance, since it allows program execution to begin with interrupts disabled. The initialization routines may then be executed prior to unmasking the interrupts.

4. The ADS uses DMA requests to refresh the CRT display and dynamic RAM. There will be gaps in the operation of the microprocessor while these requests are serviced. If the user is concerned about absolute system timing, it will be necessary to use static RAM and the "R" parameter with the Mode command to inhibit DMA requests during user program execution.

VERIFY BASIC OPERATION OF USER SYSTEM

Users should now execute a simple program to verify operation of the Emulator in their system. The clocks and control lines should be switched to obtain input from the user system. Simple programs that can be easily analyzed for proper operation are:

1. One-instruction programs that jump back to themselves.

2. I/O exercisers that read or write to a single port continuously (these are two instruction programs that read or write to an I/O port and then jump back to themselves).

3. Memory exercisers that read from, or write to a single memory location and allow the user to scope a chip select line. The use of an I/O or memory write is particularly convenient, since the microprocessor write strobe may be used as a scope sync to verify other signal lines.

If users are unable to execute simple programs of this form, these items should be checked:

1. noisy input control lines
2. ground loops
3. proper clock signals
4. proper operation of the input control lines

DIAGNOSTIC PROGRAM EXECUTION

The designer should now run very simple diagnostic programs; for example, repetitive input/output commands using a two-instruction loop. This allows the designer to probe various points in the system and verify the occurrence of decoded I/O commands. As a next step, the programs can be altered to send all combinations of data to output ports. The programs can be further modified to read an input port and write the results to an output port. These programs should be done individually for single input and output ports, to easily verify results by interrupting the process and examining the ADS register display.

The extent of these diagnostics depends on the complexity of the user I/O system. Simple parallel input and output registers can be tested by loading bit patterns into the accumulator, transferring them into the output registers, and by reading various bit patterns from the input registers. As more complex I/O functions are done, more complex diagnostic programs should be written. Therefore, it is not uncommon for a complicated peripheral device such as a floppy disk to require 10 to 20 pages of diagnostic program.

Correct operation of RAM on the user prototype should be verified at this time. Simple programs that write the low-order eight address bits into memory can be used to test the data lines and low-order address lines. More sophisticated diagnostics are needed for full verification of user memory.

USER PROGRAM EXECUTION USING ROM/PROM SIMULATOR MEMORY

The designer is now able to execute the system program, or at least parts of it, in ROM/PROM simulator memory. The system should behave as though it were operating in its final form.

The following precautions should be observed:

1. Leave the debugging vectors unchanged as long as possible. The vector locations are listed in section **Z80 Emulator Features.**

2. Be aware of the use of memory space by the Debugger. The space available to the user is shown in section **Considerations Prior to Use of Z80 In-Circuit Emulator.** Note: The user's own memory can be in the same address space as the Debugger. This memory must be installed in the user system and cannot be ROM/PROM simulator memory.

3. The relocatable Assembler and Linker can be used to put the user program at any arbitrary location. This is convenient while the program is in ROM/PROM simulator memory.

USER PROGRAM EXECUTION USING TARGET SYSTEM MEMORY

Assuming that the user memory system has been tested, the next step is to load the full program into user memory. If the program is in ROM or PROM, then the EPROMs should be programmed and installed in the user prototype. If the program is in RAM, it can be loaded into user RAM from the disk.

The U (USERMAP) command can be used to direct the required memory cycles to the user system rather than to the ADS, to allow access to user memory.

The hardware breakpoint and Logic Analyzer breakpoints are especially helpful during this phase, because a breakpoint can be set without altering the program contents. It is easier to use the debugging tools provided if the ADS debugging vector can be incorporated into the user's program.

SYSTEM OPERATION WITH MICROPROCESSOR INSTALLED

The final step in the process is to remove the Emulator probe plug from the prototype board, and replace it with the Z80 microprocessor. If the previous steps have been done carefully the system should run immediately, just as it did with the Emulator plugged into the microprocessor socket.

CONSIDERATIONS PRIOR TO USE OF Z80 IN-CIRCUIT EMULATOR

1. Output port X'F6' is reserved for switching the emulator control functions, and is not available to the user.

2. The user cannot have ROM simulator memory from X'D600' to X'D7FF'. This is used by the keyboard input routine as a work area. This area can be mapped into the user system.

3. The Debugger requires 256 bytes of memory located on an even 256-byte boundary for DEBG256. This memory must always be there and cannot be write-protected. The user program must NOT attempt to write into DEBG256. The default location is at X'D400' to X'D4FF', but DEBG256 may be relocated to any user free area.

4. The Debugger must always have a jump to DEB256+0B at location X'66' for hardware breakpoint, single-step, trace, Logic Analyzer and BREAK key. It requires a similar jump vector at one of the eight restart locations (0-7) for software breakpoints.

5. The single-step and trace functions can operate with all maskable interrupts disabled by first executing an N command. This will disable interrupts on all following single-step, trace, or N commands. Execution of an E command will cause all subsequent single-step, trace or E commands to start with interrupts enabled.

6. The output lines of the Z80 are in the following states during the three emulator modes:

	Emulation User Memory Access	Emulation Host Memory Access	Debug
Address 0-15	Active	Active	Active
Data 0-7	Active	Active(1)	Tri-state
MREG-	Active	Active	Inact(H)
IOREG-	Active	-	Inact(H)
M1-	Active	Active	Active
BUSACK-	Active	Active	Inact(H)
RD-	Active	Active	Inact(H)
WR-	Active	Inact(H)	Inact(H)
HALT	Active	Active	Active
RFSH	Active	Active	Active

7. The Stack Pointer must be pointing to an area of RAM before execution begins, and at any time when a breakpoint occurs.

NOTE

This feature can be used to reset Z80 peripheral chips by sending them the RTI instruction fetched from the host ROM simulator. This is only possible if ALL USER BUS DRIVERS are INACTIVE for host memory accesses.

Control Lines and Their Effect

1. Clock line: Problems will show up when using the MC command.

2. RST-: Active low, if active will force processor to location X'00', and will not let user program proceed until reset is cleared.

3. INTR-: Active low, if active will force processor to execute interrupt upon entering the user program.

4. NMI-: Active low, if active will force processor to execute Non-Maskable Interrupt prior to proper entry to the user program.

5. Ready: Active low, means a wait condition. If it does not go high, it will lock up the processor.

6. BUSRG-: Active low, means a DMA request. If it does not go low, it will lock up the processor.

To provide an Emulator that is a close approximation of the Z80 Microprocessor, it is necessary to provide minimum isolation between the user system and the In-Circuit Emulator logic. Thus, there are no terminations on the data lines, address lines, control lines and clock lines at the Emulator probe connections. The Emulator will be sensitive to noise on the data and control lines of the user system, just as the microprocessor chip would be.

Many of the problems associated with microprocessor-based systems are associated with noisy data and control lines. If TTL bus drivers are used to buffer the data bus, noise can be generated when the data bus is turned around to switch from Read to Write. Schottky TTL chips such as 8T97, 8T98, 8T26, 8216, 8226, 74LS240, 74LS244 and 74LS245 are capable of generating very narrow switching transients (less than 5 nanoseconds). These can cause a significant amount of crosstalk between data and control lines when the data lines switch. In addition, control signals used to turn around bi-directional data buses may generate bus conflicts at the instant directions are changed. This can also cause significant crosstalk.

Another caution concerns the connection of the grounds between the ADS and Emulator, and the user system. The Emulator logic ground is connected to the ADS frame and to the third wire on the power cord at a single point on the system power supply. If the user system also has a third wire connection there will be a ground loop through the power line that may generate considerable ground noise. This condition can be avoided by disconnecting the user logic ground from the user chassis during the Emulator debugging phase.

There are three distinct Emulator modes. When the Emulator is running the user program and accessing user memory, the output lines closely match the chip characteristics. When the Emulator is running the user program but accessing ROM simulator memory, some of the control lines are inactive in order to access ROM simulator memory rather than user memory. When the processor is executing the Debugger, the address and data lines are active while the control lines are inactive.

The user needs to be aware of the location of the stack. The Debugger sets the stack pointer to a Read/Write area in the ADS memory. This stack will be used both by the user software and by the Debugger to save and restore the registers at exit from, and entry to the user program. If this memory location is mapped into the user space, it must still be Read/Write memory, otherwise the user program must load the stack pointer with a suitable address prior to execution. The Debugger will not operate correctly without a stack in Read/Write memory.

INSTALLATION

Installation of the Z80 In-Circuit Emulator involves the following steps:

1. Remove jumper connectors D, G, and J on the Z80 CPU card to allow gating of memory requests and clock signals via the Emulator interface card. If the system is equipped with high-speed static RAM, the jumper at A should be installed. This deletes a one-cycle wait state on all Z80 M1 cycles mapped into the ROM/PROM simulator memory. This wait state is necessary when using dynamic RAM at 4Mhz. Jumper E selects 2Mhz clock; Jumper F selects 4Mhz clock.

2. The short 40-pin cable (Part No. 2300-0303) is used to connect J1 on the Z80 CPU card and J3 on the Emulator interface card.

3. After connecting the Z80 CPU card (Part No. 2300-4711) and the Emulator interface card (Part No. 2300-4712), place the two cards on the table face up with the edge finger connectors to the left. The Emulator interface card will be on top. Place the Emulator probe assembly (Part No. 2300-4018) on the table with the probe connector to the right. Connect the topmost cable from the probe assembly to J1 (the lower or leftmost of the two connectors) on the Emulator interface card. Connect the lowermost cable from the probe assembly to J2 (the upper or rightmost of the two connectors) on the Emulator interface card. Refer to **Figure 2-1** for details.

4. Install the Z80 CPU card and Emulator interface card in the ADS mainframe in adjacent slots.

5. The two cables connecting the Emulator probe assembly and the Emulator interface card can be folded and brought out the rear of the ADS through one of the cable clamps at the bottom of the ADS case.

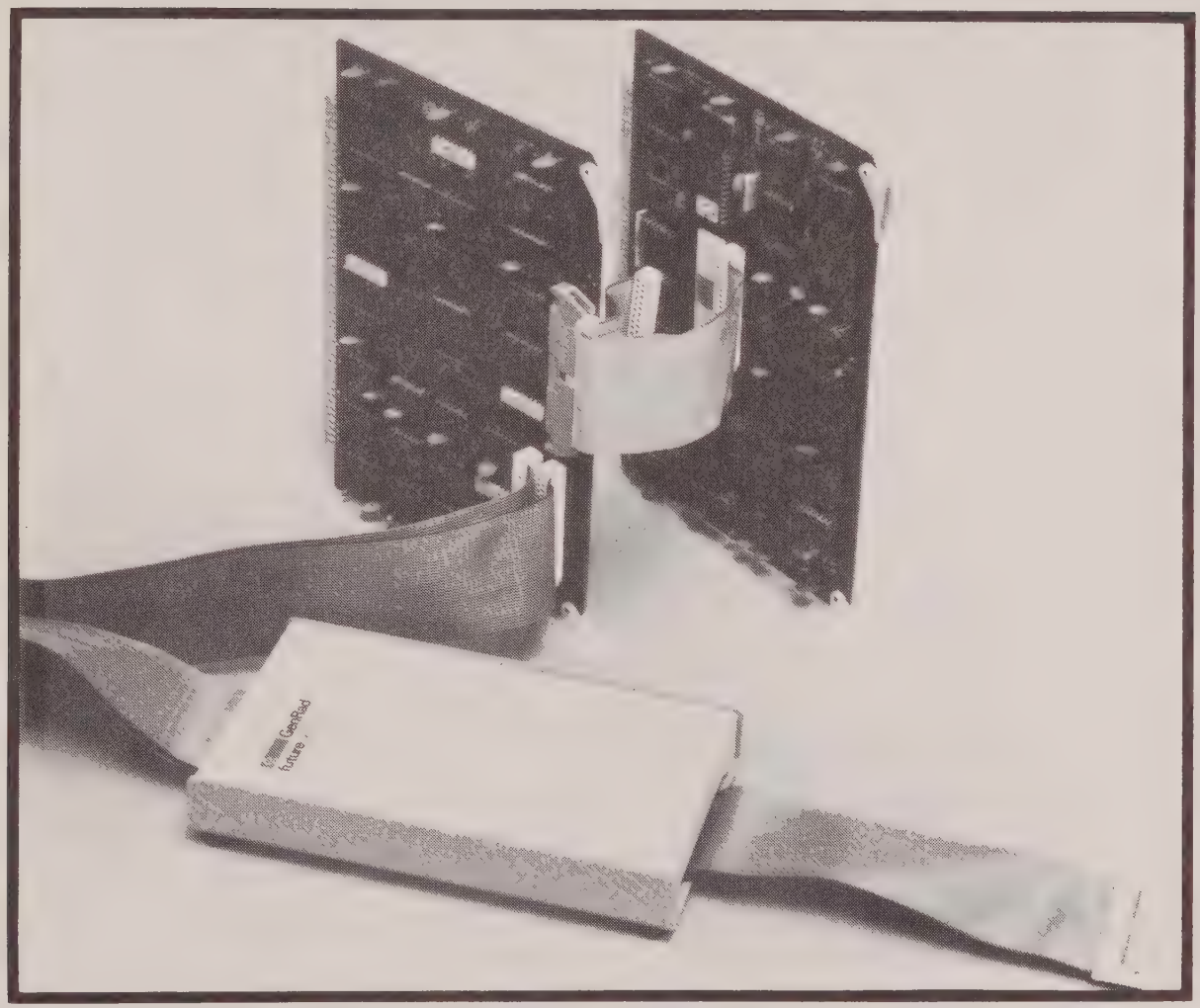


Figure 2-1. Z80 In-Circuit Emulator Assembly

Z80 PROBE - PHASING ADJUSTMENT PROCEDURE

EQUIPMENT REQUIRED

Z80 Emulator-Equipped 2300 System

2300-4018 Adjustable Emulation Probe

> 50 Mhz Dual Channel Oscilloscope

2 Oscilloscope Probes (10 Megohms)

1 Phillips screwdriver

1 small screwdriver or adjustment tool

PROCEDURE

STEP 1 - Power down the system and disc drive.

STEP 2 - Install a 50 Mhz scope, dual channel equipped with two 10 Megohms scope probes.

STEP 3 - CHECK:

Remove your system CPU board (part number 2300-4711). U12 is a Z80A chip, or MOSTEK Z80-4 (4Mhz capability)

JUMPERS IN = A , F

JUMPERS OFF = D , J , G , E

STEP 4 - Open the probe box by unscrewing the four rubber feet and removing the cover.

STEP 5 - Insert the Emulation probe tip in the Z80 socket (prototype board) with a pin extension feature.

STEP 6 - Power up the ADS and target (prototype) system.

STEP 7 - Set Scope

- time base at: 50 NS / DIV.
- chopped (not alternate)
- trigger mode = internal

STEP 8 - Connect channel A scope probe at U1 Pin 13 of Emulator probe.

STEP 9 - Turn potentiometers R2 and R4 of Emulator probe fully CW. (about 20 turns).

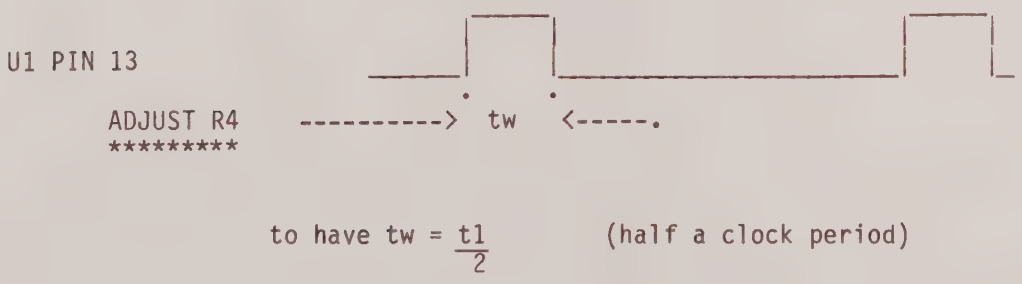
STEP 10 - Connect channel B scope probe at Pin 6 (clock) of Emulator probe, trigger on channel B. Lock channel B.

** [MEASURE AND NOTE t1 THE EXACT PERIOD OF THE CLOCK
[
[
[(from falling edge, 4.75 V to falling edge).
[
[
[
[
[EG: IF Fclk= 4Mhz T1 = 250 NS.
[

See diagram **Figure 2-3.**

STEP 11 - Now set trigger source to channel A. Set trigger slope to positive.

STEP 12 - Adjust potentiometer R4 (CCW) to have on channel A:



to have tw = $\frac{t1}{2}$ (half a clock period)

Figure 2-2. Adjust Potentiometer R4

STEP 13 - Move probe channel A to Emulator probe pin 19.

(Pin 19 = MREQ NOT)

STEP 14 - Type " JD " on ADS keyboard (call Debugger)

Type " MC " on ADS keyboard, (Map Clock)

(If the system fails, repeat procedure from Step 11, else continue.)

STEP 15 - Set trigger slope negative on MREQ (channel A)

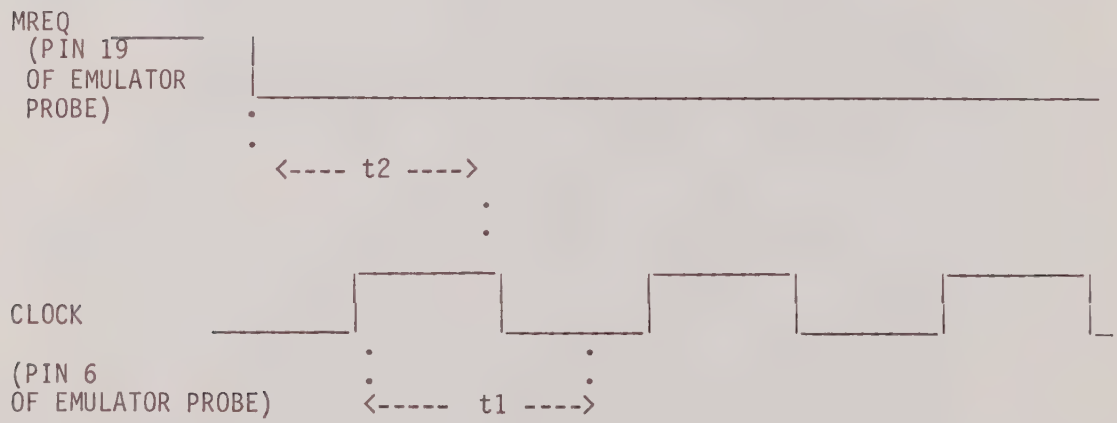
STEP 16 - Calculate t2 = t1 - 40NS.

(for above example t2 = 250 - 40 = 210 NS

STEP 17 - Adjust potentiometer R2 to have:

t2 (previously calculated as explained at Step 15) between falling edge of MREQ NOT and next falling edge of the clock.

Examples: see **Figure 2-3.**



ADJUST R2 : t2 = t1 - 40 ns.
****

EG : if clock = 2Mhz t2 = 500-40=460Ns

Figure 2-3. Adjust Potentiometer R2

When this adjustment is completed, both potentiometers may be fixed with a drop of glyptol.

Chapter 3

UNIVERSAL LOGIC ANALYZER

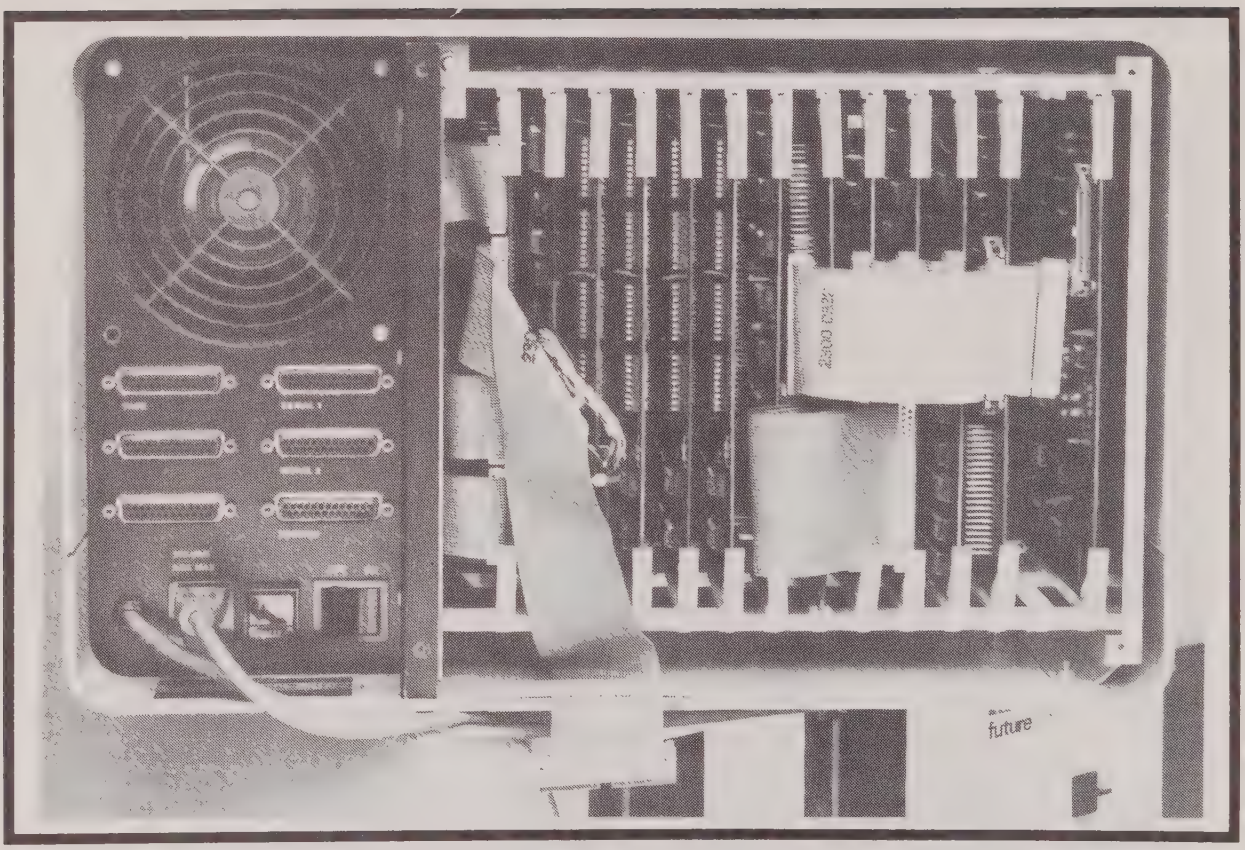
INTRODUCTION

The Universal Logic Analyzer greatly simplifies troubleshooting and debugging tasks by providing a means to view bus data during program execution. The Logic Analyzer gathers a maximum of 256 traces for up to 16 address lines, up to eight data lines, the Read/Write line, the I/O line, the DMA line, the M1 line and four external user-defined lines. Information is recorded synchronously with the system or prototype clock to 4 MHz and stored in an area of memory contained in the Logic Analyzer hardware.

The Trace Qualifier register selects the traces to be stored in Logic Analyzer memory. The type of information to be stored can be restricted, such as a READ or WRITE to a particular location. Three breakpoints are provided to halt data storage and optionally to halt program execution. The breakpoints may be ANDed or ORed. In addition, a count may be set for each breakpoint so that the breakpoint conditions will be satisfied only when the conditions have occurred n number of times. A delay count may be entered as an element of the Trace Qualifier to continue storage for up to 65,535 qualified bus cycles after the breakpoint conditions have been met.

Recorded information may be displayed in either a parallel state (cycle-by-cycle) format or a serial timing diagram (line-by-line) format. The parallel state display shows 16 traces at a time in hexadecimal form. Each trace represents the values of all 32 lines during one bus cycle. The serial timing diagram display shows the logic states of up to 16 selected lines in square wave form. Each line may contain up to 64 states. Special function keys and commands may be used with either form of display to step through all 256 traces.

The Logic Analyzer package consists of four printed circuit cards and an external probe assembly. The cards plug into the 2300-ADS card cage and are interconnected with a series of cables.



NOTE	Slot	Designation
Card positions are referenced with observer facing the rear of the console. Position 1 is the leftmost slot; position 14 is the rightmost slot.	1	MPIO
	2	CRT
	3	RAM 16K
	4	RAM 16K
	5	RAM 16K
	6	RAM 16K
	7	Z80 CPU
	8	LOGIC ANAL INT
	9	LOGIC ANAL A
	10	LOGIC ANAL B
	11	LOGIC ANAL C
	12	Z80 EMULATOR
	13	SPARE
	14	DEBUG/PROM

Figure 3-1. ADS Backview

LOGIC ANALYZER ENTRY AND EXIT

To invoke the Logic Analyzer from the Debugger display, type

A<RETURN>

To return to the Debugger, type the D command in either of the following ways:

D{expression}	expression consists of a constant, an indirect address, a relative address, a symbolic address or a combination of such values; expression sets a memory pointer;
D$	return to previous display;
D	returns to the Debugger at the Eaddr (execution address).

LOGIC ANALYZER DISPLAY

Upon entry to the Logic Analyzer, four major parameters are displayed on the CRT screen as shown in **Figure 3-2.** These parameters are described below.

BREAKPOINTS

The Logic Analyzer provides three breakpoints labeled B0, B1, and B2. Program execution and data storage stop when conditions specified in the breakpoints have been met. Allowable breakpoint parameters are listed in **Table 3-A.**

AND/OR PROVISION

The AND mode of operation requires that the designated breakpoint conditions be satisfied in sequence before the Logic Analyzer will stop data storage. For example, B1 conditions cannot be satisfied before B0 conditions have been met.

The OR provision causes the Logic Analyzer to halt the storage process when any one of the three breakpoint conditions has been met.

TRACE QUALIFIER

The Trace Qualifier (Q) specifies the characteristics of the traces to be stored in Logic Analyzer memory. The delay count parameter causes the Logic Analyzer to continue storing data for a specified number (up to 65,535) of bus cycles after the breakpoint conditions have been met. Allowable Trace Qualifier parameters are listed in **Table 3-B.**

TIMER INCREMENT

The 24-bit timer measures the time elapsed between satisfaction of breakpoints B0 and B1. Three types of measurement units may be selected by the commands listed below:

TN	specifies .1 microsecond (100 nanoseconds)
TU	specifies 1 microsecond
TB	specifies 1 bus cycle

DEFAULT CONDITIONS

Default conditions for the above parameters are:

B0 Break immediately on all but DMA cycles

B1 These breakpoints are not examined
B2 because Count (C) equals 0

Q Store all traces except DMA

T Times in bus cycles

The Trace Qualifier line and each of the breakpoint lines may be reset to their respective default conditions by typing the line name followed by return. For example;

B0 <RETURN>

AND	COUNT	ADDR	DATA	IO	RW	DMA	M1	EXTR	PC=0000	Eaddr=0000
B0	1	=xxxx	xx	x	x	0	x	xxxx		
B1	0	=xxxx	xx	x	x	0	x	xxxx	Timer=0	cycles
B2	0	=xxxx	xx	x	x	0	x	xxxx		
Q	0	=xxxx	xx	x	x	0	x	xxxx		

Figure 3-2. Logic Analyzer Parameters Default Values

TABLE 3-A. BREAKPOINT PARAMETERS

PARAMETER	CONDITION	EXPLANATION
COUNT	=0	Breakpoint immediately satisfied.
	=1 to 65535	Number of times the breakpoint conditions must be met in order to satisfy the breakpoint.
ADDR	=4 hex characters.	
	=#SYMBOL	Symbolic debug mode.
	=XXXX	Don't care. Hex characters and "X" can be mixed, e.g.: ABCE X123 X1X1
	= expression	Break when A equals any allowable address.
	>= expression	Break when A is greater than or equal to expression. "X" is not allowed.
	<= expression	Break when A is less than or equal to expression. "X" is allowed.
DATA	=2 hex characters	
	=XX	Don't care. Hex characters and "X" can be mixed, e.g.: AB AX 1X
IO	=1	Break on I/O execution.
	=0	Break on other than I/O execution.
	=X	Don't care.
RW	=R	Break on Read cycle (memory or I/O).
	=W	Break on Write cycle (memory or I/O).
	=X	Don't care.
DMA	=1	Break on DMA cycle.
	=0	Break on other than DMA cycle.
	=X	Don't care.
M1	=1	Break if M1 bus line is high (instruction fetch).
	=0	Break if line is low.
	=X	Don't care.
EXTR	=4 bits	Store depending on the status of the four external lines.
	=XXXX	Don't care. Binary and 'X' characters may be mixed. For example: XX01 11XX 1XX0

TABLE 3-B. TRACE QUALIFIER PARAMETERS

PARAMETER	CONDITION	EXPLANATION
COUNT	=0	No qualification of stored data.
	=1 to 65535	Number of qualified cycles to be stored after breakpoint conditions are satisfied (DELAY).
ADDR	=4 hex characters	
	=# symbol	Symbolic debug mode.
	=XXXX	Don't care. Hex characters and "X" can be mixed. For example: ABCE X123 X1X1
	= expression	Store when A equals any allowable address.
	>= expression	Store when A is greater than or equal to expression. "X" is not allowed.
	<= expression	Store when A is less than or equal to expression. "X" is not allowed.
DATA	=2 hex characters	
	=XX	Don't care. Hex characters and "X" can be mixed For example: AB AX 1X
IO	=1	Store on I/O execution.
	=0	Store on other than I/O execution.
	=X	Don't care.
RW	=R	Store on Read cycle (memory or I/O).
	=W	Store on Write cycle (memory or I/O).
	=X	Don't care.
DMA	=1	Store on DMA cycle.
	=0	Store on other than DMA cycle.
	=X	Don't care.
M1	=1	Store if M1 bus line is high (instruction fetch).
	=0	Store if line is low.
	=X	Don't care.
EXTR	=4 binary characters	Store, depending on the status of the four external lines.
	=XXXX	Don't care. Binary and 'X' characters may be mixed. For example: XX01 11XX 1XX0

EXAMPLE: Breakpoint and Trace Qualifier

Input the following program using the Editor:

```
                RSEG     TEST
START           INC      A
                OUT      (X'00'),A
                JP       START
                END      START
```

Write the program to a file using the W command:

```
W{filename1}
```

Jump to the Assembler by entering:

```
JA <RETURN>
```

Enter the following information:

```
SPECIFY ASSEMBLER OPTIONS:

(L) - LIST ON SCREEN
(T) - TRUNCATED LIST
(S) - INCLUDE SYMBOL TABLE
(E) - LIST ERRORS ONLY
(Z) - FLAG NON-8080 OPCODES
>LS <RETURN>

SOURCE FILE:
>{filename1}    <RETURN>

MACRO FILE:
>               <RETURN>

OUTPUT FILE:
>{filename2}    <RETURN>

LISTING FILE:
>               <RETURN>
```

When <RETURN> is entered, the Assembler starts processing and the following display is shown on the screen:

```
PASS 1
PASS 2
                RSEG     TEST
0000  3C        START    INC  A
0001  D3 00              OUT  (X'00'),A
0003  C3 0000            JP   START
0006                     END  START
START           0000     TEST 6 0000
NO ERRORS
```

Depress <RETURN> to get the cursor back on the screen and jump to the Linker by entering:

```
      JL<RETURN>
```

```
SPECIFY LINKER OPTIONS:
(D) - DELETE UNSPECIFIED RSEGS
(S) - INCLUDE SYMBOL TABLE
(L) - LIST TABLES ON CRT
>SL              <RETURN>

SPECIFY INPUT FILE:
>{filename2}     <RETURN>
RSEG    LENGTH
TEST    0006

SPECIFY INPUT FILE:

SPECIFY OUTPUT FILE:
>{filename3}     <RETURN>

SPECIFY LISTING FILE:
>                <RETURN>

LINKER INPUT:
> #ORG X'0100'   <RETURN>
>                <RETURN>
```

When <RETURN> is entered, the Linker starts processing and the following display is shown on the screen:

```
FILE  RSEG      ADDR     LENGTH

ENTRY POINT 0100
FUNCTION COMPLETED
```

Enter <RETURN>. When the cursor appears on the screen, invoke the Debugger by entering:

```
      JD<RETURN>
```

Load the assembled linked program at location 100H with the symbol table at location 108H as follows:

```
      L{filename3} ,108
```

The message SYMTAB LENGTH = 0016 appears on the message line to specify the length of the symbol table.

Enter D100 <RETURN> on the command line to display the program loaded at 100H and the symbol table loaded at 108H. The disassembly display for the program appears as follows:

```
0100  3C        INC     A
0101  D300      OUT     (X'00'),A
0103  C30000    JP      X'0100'
```

Invoke the Logic Analyzer to experiment with breakpoints and trace qualifiers by entering the following command:

A <RETURN>

DESIRED BREAK CONDITION	LOGIC ANALYZER COMMAND
Output X'AA'	B0,A=AA00,D=AA,IO=1,RW=W
100 times past JP instruction	B0,C=100,D=C3,M1=1
Time in microseconds between output of X'00' and X'FF'	B0,A=0000,D=00,IO=1 B1,C=1,A=FF00,D=FF,IO=1,RW=W TU
Break if go outside program;	0 B0,C=1,A<=FF,M1=1 B1,C=10,A>=106,M1=1 B2,C=1,A<=FF,M1=1

Execute each step by typing E100 <RETURN> (starting address of the program).

EXECUTING A PROGRAM WITH LOGIC ANALYSIS

To begin Logic Analyzer operation, the user must select the Debugger (JD), load the user program (L command), and invoke the Logic Analyzer (A command). The CRT screen will display the default information shown in **Figure 3-2.** Each hexadecimal digit in the ADDR and DATA columns represents four bits (0-F) and the information in each remaining column represents one bit.

After setting the parameters for a particular need, the user program may be executed using an E or an N command from the Logic Analyzer.

Execution from Logic Analyzer

A program may be executed from the Logic Analyzer in one of six ways:

E{expression}	expression is any allowable combination of addresses or symbolic references;
E$	execution begins at the current program counter (PC=);
E	execution begins at the pointer ">" on the active side of the last Debugger display (Eaddr=);
N{expression}	expression is any allowable combination of addresses or symbolic references (interrupts are disabled);
N$	execution begins at the current program counter; (interrupts are disabled);
N	execution begins at the pointer ">" on the active side of the last Debugger display (interrupts are disabled).

When program execution is begun in Logic Analyzer display mode, both data storage and program execution will halt when the Logic Analyzer breakpoints have been satisfied. The Logic Analyzer display will return to the screen with the flashing message:

Break conditions satisfied

The "BREAK+0" message in reverse video represents the break trace line.

If the breakpoints were not satisfied, the Logic Analyzer stops data storage several instructions after program execution has stopped. These last instructions are operating system instructions which interrupt the system, save the user's program status and issue an OUT instruction to port X'AF' which halts the Logic Analyzer Storage process. The Debugger display is returned to the screen and the breakpoint parameters are set back to default conditions.

The user can switch between the Logic Analyzer and the Debugger. If the Logic Analyzer display is on the screen, enter D<RETURN> on the command line to invoke the Debugger. To return to the Logic Analyzer display, enter A<RETURN> on the command line.

INTERPRETATION OF RESULTS

Two display formats are available to users of the Logic Analyzer: a parallel state (hexadecimal) display, and a serial (timing diagram) display. The parallel state display is most useful when comparing a program listing with program execution history stored by the Logic Analyzer. The serial display is most useful in determining the timing relations between signals, such as a data line and an external line.

PARALLEL STATE DISPLAY

The parallel state display shows 16 traces at a time in hexadecimal form. Each trace represents the values of the 32 lines during one bus cycle. As shown in **Figure 3-3**, the screen is split with the left half displaying the disassembled program and the right half displaying the bus data.

The up and down arrow keys may be used to step through all 256 traces. To move up or down quickly, the P command may be used. For example:

P+5 moves ahead 5 lines
P-10 moves back 10 lines

The BREAK $\pm$ indicator on the reverse video line shows the number of qualified bus cycles between the reverse video trace and the trace which caused the break conditions to be satisfied (or the last trace stored if the break conditions were not satisfied). The break trace line is highlighted.

```
AND       COUNT     ADDR      DATA      IO   RW   DMA  M1   EXTR   PC=0000
Eaddr=0000
B0        1         =XXXX     XX        X    X    0    X    XXXX
B1        0         =XXXX     XX        X    X    0    X    XXXX   TIMER=0
cycles
B2        0         =XXXX     XX        X    X    0    X    XXXX
Q         0         =XXXX     XX        X    X    0    X    XXXX


OPCODE    MNEMONIC  OPERAND         ADDR      DATA IO   RW   DMA  M1   EXTR
C33E74    JP        X'743E'         0038      C3   0    R    0    1    1111
                                    0039      3E   0    R    0    0    1111
                                    003A      74   0    R    0    0    1111
E5        PUSH      AF              743E      E5   0    R    0    1    1111
                                    6FF9      00   0    W    0    0    1111
                                    6FF8      42   0    W    0    0    1111
3A0F88    LD        A,(X'880F')     743F      3A   0    R    0    1    1111
                                    7440      0F   0    R    0    0    1111
.BREAK-8
                                    7441      88   0    R    0    0    1111
                                    880F      00   0    R    0    0    1111
F606      OR        X'06'           7442      F6   0    R    0    1    1111
                                    7443      06   0    R    0    0    1111
D3F6      OUT       (X'F6'),A       7444      D3   0    R    0    1    1111
                                    7445      F6   0    R    0    0    1111
                                    06F6      06   1    W    0    0    1111
                                    7446      D3   0    R    0    1    1111
```

Figure 3-3. Parallel Display

SERIAL(TIMING DIAGRAM) DISPLAY

The timing diagram is constructed from the data stored in the Logic Analyzer and uses four special graphics characters to produce the timing diagram:

- high
_ low

_|‾ rising knee

‾|_ falling knee

The eight bits of information representing the status of one line are examined and the appropriate symbol (high or low) is produced for each bit until a change of state is detected. For example, the eight-bit string 11011100 appears in the timing diagram as:

1 1 0 1 1 1 0 0

Note that the signals change (high or low) between the bits.

The up, down, left and right arrow keys may be used to step through all 256 traces. To quickly move left or right across the screen, the P command may be used. For example:

P+5 moves 5 traces to the right;
P-10 moves 10 traces to the left

To view one or more lines in non-sequential order, enter the FS command followed by the desired line number(s). For example:

FS,A02,D1,RW

moves the A02, D1 and RW lines to the top of the display. FS, A15 returns the Logic Analyzer to the default display.

The BREAK + indicator on the reverse video line shows the number of qualified bus cycles between the reverse video trace and the trace which caused the break conditions to be satisfied (or the last trace stored if the break conditions were not satisfied).

The timing diagram is continuous only if the Logic Analyzer is storing all machine cycles. Therefore, all Trace Qualifier (Q) parameters which affect storage must be set to "X" (don't care). The bus cycle which satisfied the break conditions is a BREAK+0.

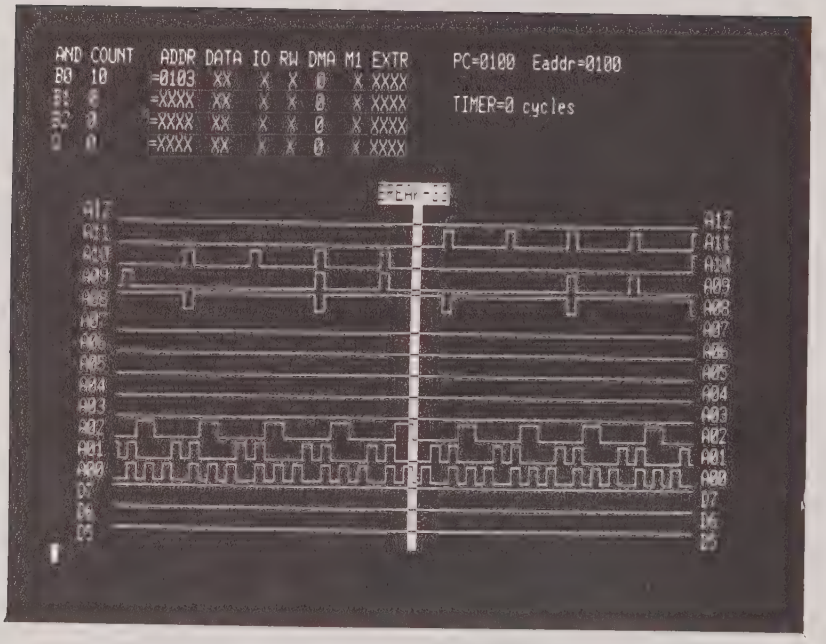

AND COUNT ADDR DATA IO RW DMA M1 EXTR
B0 10 =0103 XX X X 0 X XXXX
=XXXX XX X X 0 X XXXX
=XXXX XX X X 0 X XXXX
=XXXX XX X X 0 X XXXX
PC=0100 Eaddr=0100
TIMER=0 cycles

Figure 3-4. Serial Display

LOGIC ANALYZER INSTALLATION

The Logic Analyzer consists of three Analyzer cards, one Analyzer Interface card, and one probe assembly labeled as follows:

Analyzer A is labeled 2300-4730 under J1.
Analyzer B is labeled 2300-4731 under J1.
Analyzer C is labeled 2300-4732 on the left edge of the card.
The Analyzer Interface card is labeled 2300-4753 on the reverse side.
The probe assembly is labeled 2300-4008 with an adhesive label.

Install these cards according to the steps below:

1. Place the three Analyzer cards in consecutive order (A, B, and C) from right to left when looking at the front of the ADS. Attach the 4-inch, 3-connector 40-pin cable (Part No. 2300-0321) to J2 on each card.

2. Place the Analyzer Interface card in front of Analyzer A and attach J4 on the Analyzer Interface card to J1 on Analyzer A with a shorter 2-connector 50-pin cable (Part No. 2300-0306).

3. Use the longer 2-connector 50-pin cable to connect J1 on the Analyzer Interface card to J1 on Analyzer B (Part No. 2300-0305).

EXTERNAL PROBE ASSEMBLY CONNECTION

Connect the 20-conductor cable on the External Probe Assembly to J6 on the Analyzer Interface card.

NOTE

The arrows on the connectors must be aligned.

CPU AND EMULATOR INTERFACE CARD CONNECTION

Position the CPU card and the In-Circuit Emulator Interface card in front of the Logic Analyzer card set. Connect the Z80 CPU and Emulator cards by the longer 3-connector 40-pin cable to J2 on the Analyzer Interface card. The 3-connector 40-pin cables replace the 2-connector 40-pin cables used to connect the CPU and Emulator Interface cards.

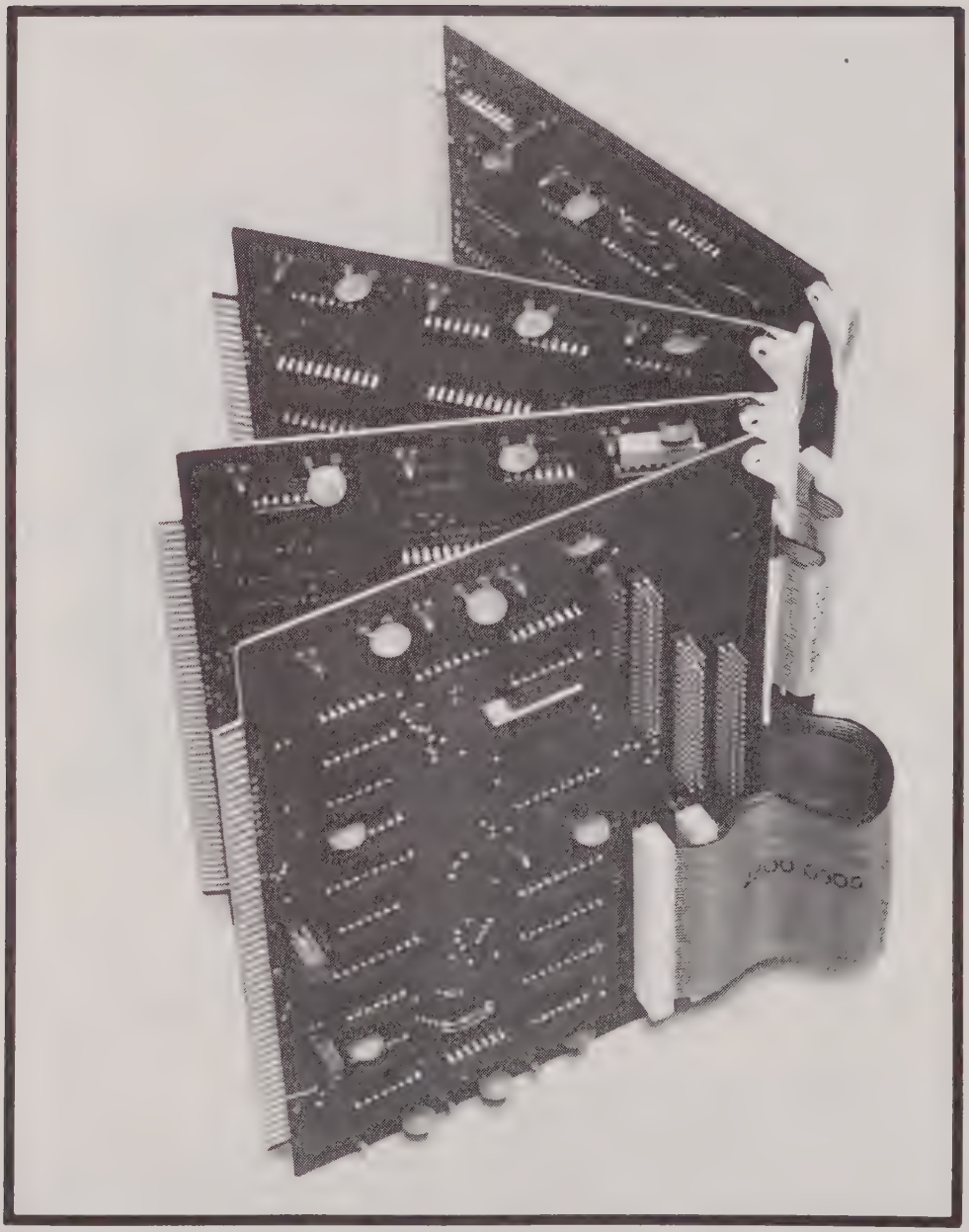


Figure 3-5. Logic Analyzer Card Assembly

STRAPPING

The Analyzer Interface strapping connections are listed in Table 3-C.

TABLE 3-C. STRAPPING CONNECTIONS

	Z80 No Emulator	Z80 Emulator
INT	X	
Emulator		X
68M		
SBM	X	X
*R/S		

*R/S indicates a RUN/STOP mode reserved for later expansion capability.

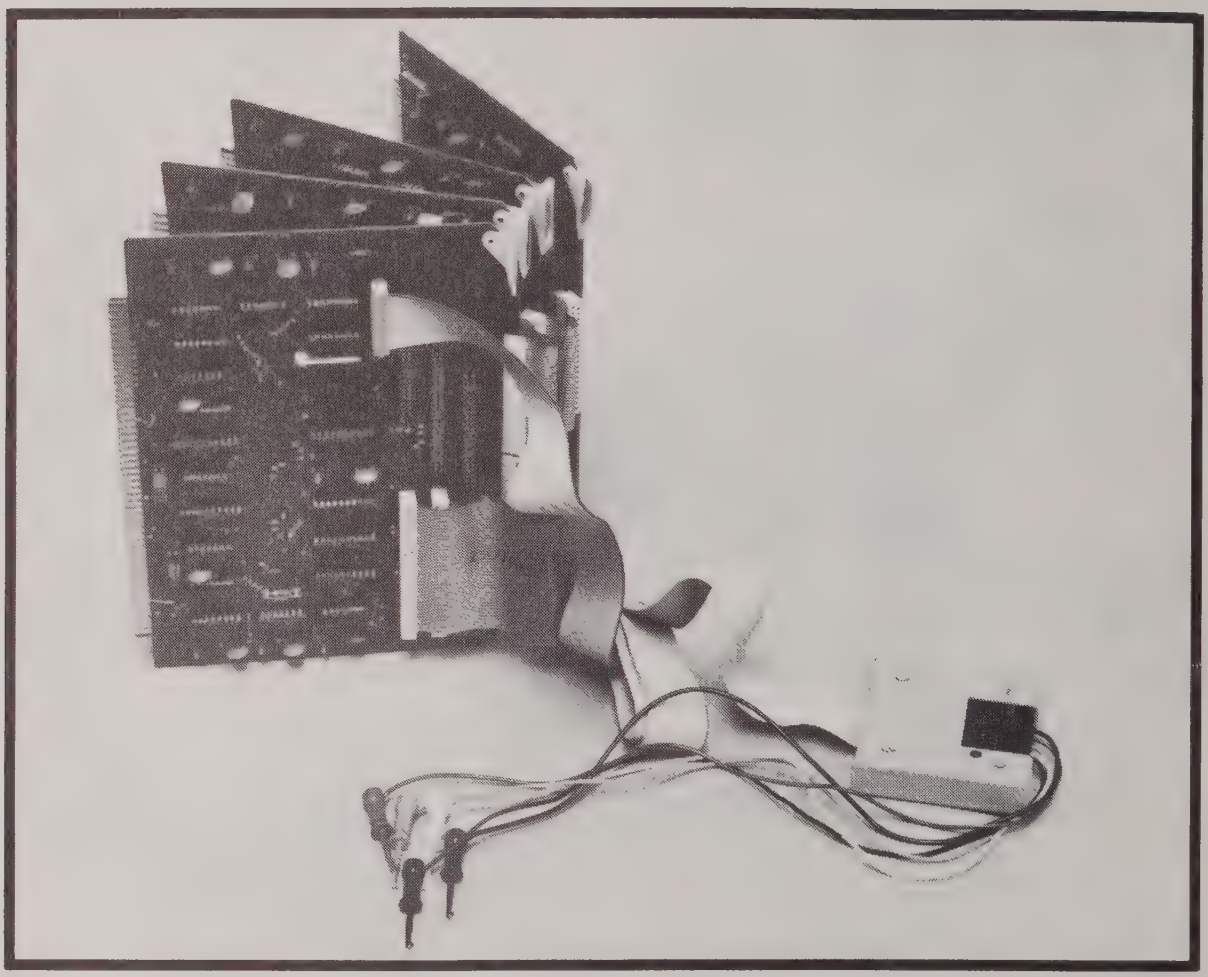


Figure 3-6. Logic Analyzer/External Probe Assembly

Chapter 4

EPROM PROGRAMMER

INTRODUCTION

The following equipment is required to use the EPROM Programmer with the 2300 Advanced Development System (ADS):

1. A DEBUG/EPROM Programmer card (Part No. 2300-4705) which plugs into the ADS card cage.

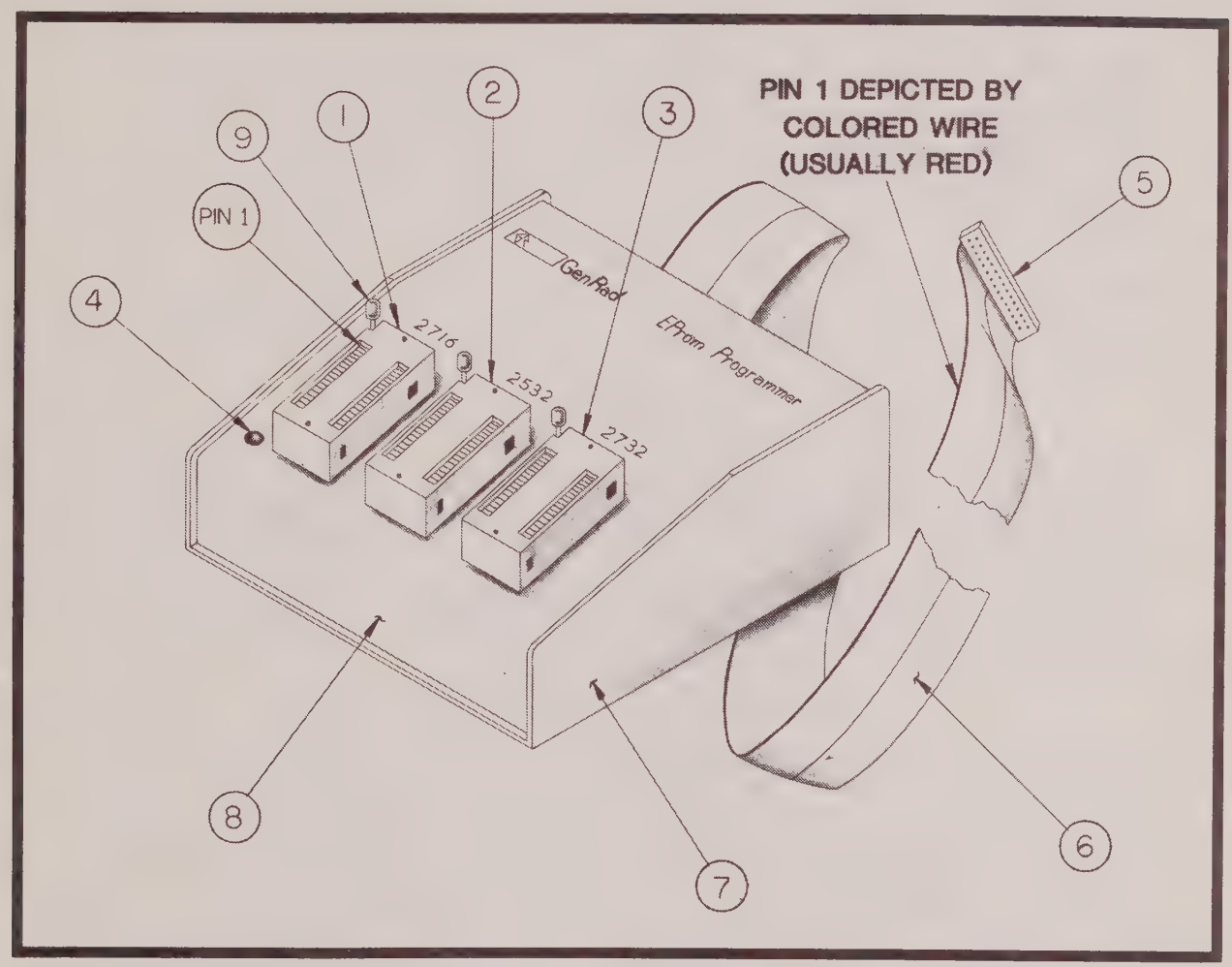
2. The EPROM Programming module for the intended EPROM. The 2704 and 2708 EPROMs are programmed on the 2708 EPROM Programmer (Part No. 2399-9010, **Figure 4-1**) and the 2716, 2516, 2732, 2532, and 2758 EPROMs are programmed on the 2716 EPROM Programmer (Part No. 2399-9009, **Figure 4-2**). The latter module has three sockets. The module connects to the EPROM Programmer card with a plug-in cable.

3. Debugger software including the EPROM programming modules RDOS II SYSTEM FILES, Part No. 2300-6096-XX, and RDOS II DEBUGGER/LOGIC ANALYZER, Part No. 2300-6098-XX). (The XX in the part numbers refers to revision levels, which may vary.) The file name is DEBUGZ80L, it must reside on the SYSTEM FILES disk in Drive 0, and it must have a Z attribute to indicate that it is a system file. It must also be the only system file beginning with the letter D.

4. A copy of the 2300 Logic Analyzer/Debugger Reference Card (Part No. 2300-5012).

5. ADS Software Reference Manual (Part No. 2300-5000-01).

Figure 4-1. 2704/2708 EPROM PROGRAMMER



1 EPROM socket to 2716/2516 EPROM
2 EPROM socket to 2532 EPROM
3 EPROM socket to 2732/2758 EPROM
4 LED indicator for V_{pp}
5 Connector to ADS
6 Interconnecting ribbon cable
7 Chassis
8 Cover
9 EPROM shift locks

Figure 4-2. 2716/2532/2732 EPROM Programmer

INSTALLATION

Power down the system. Insert the cable from the EPROM Programmer into the Debugger board (labeled DEBUG/EPROM PROGR in the top right corner of the component side of the board). The cable should extend away from the board. See **Figure 4-3** for details.

Inserting the cable backwards will result in possible destruction of the Debugger board.

OPERATION

Load the Logic Analyzer/Debugger software from the disk by entering the command:

L{filename}{,offset} JD<RETURN>

All EPROM programming commands are available in this file. Place the EPROM Programmer on a flat surface with the EPROM socket facing you, and insert the EPROM with pin 1 on the top left side closest to the ribbon cable.

Using the L command, load the EPROM image from the disk into memory. This command will load a file from the floppy or the hard disk. The L command has the format:

L filename ,offset [,symbol table address]

where filename is the name of the file containing the EPROM image and offset is a hex address that can be used to position the EPROM image in an unused portion of memory.The sum of the filename address plus the offset equals the location in memory where the EPROM image is loaded. This address will be added to the starting address of the EPROM image. In the Z80 processor, an offset of at least 100 Hex must be used if the EPROM image starting address is at 0. This will eliminate possible conflicts between the EPROM image and the interrupt vectors. Refer to **Figures 4-4 and 4-5** for memory map diagrams of 48K and 64K systems respectively.

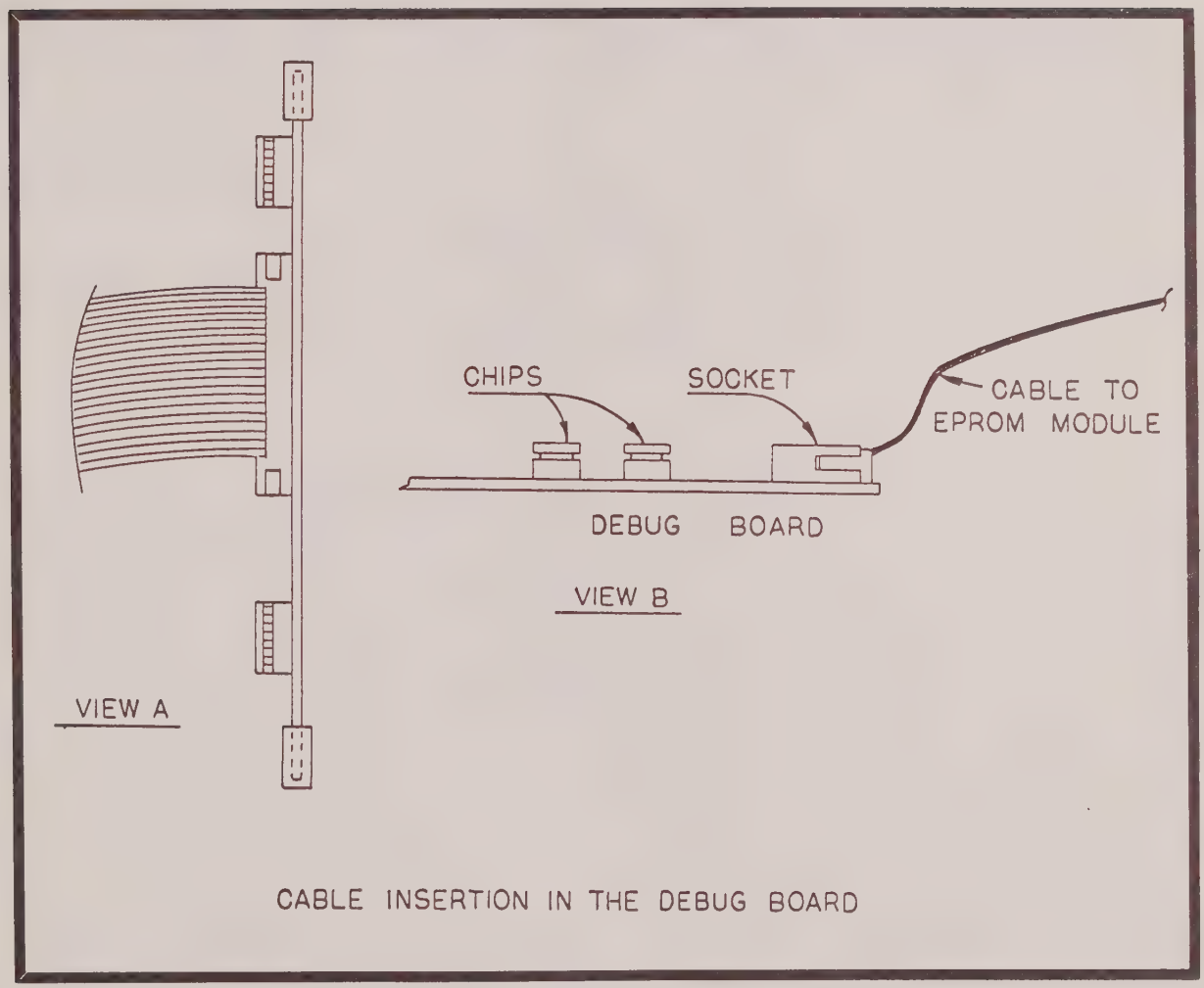


Figure 4-3. Cable Insertion In The Debug Board

Debugger memory available to the user in a 48K system is diagrammed below. An offset address must be used if the area users need for loading EPROM image is being used by the system.

Figure 4-4. Debugger Memory 48K System

Debugger memory available to the user in a 64K system is diagrammed below. An offset address must be used if the area users need for loading EPROM image is being used by the system.

Figure 4-5. Debugger Memory 64K System

PROGRAMMING AN EPROM

Place an empty EPROM in the EPROM Programmer socket. Use the Check for EPROM Empty command:

C[type][,length]

to verify that the EPROM is empty. The message EPROM EMPTY should be displayed at the bottom left side of the screen. If not, replace EPROM.

To program the EPROM, use the Program EPROM command:

P[type][,length][,interleaving factor]

The EPROM will take approximately two minutes to program. (The time it takes depends on the size of the EPROM and the area to be programmed). Some types of EPROMs require that the entire EPROM be programmed. The first byte of code to be programmed should be at the memory pointer of the active side of the Debugger.

When the programming pass is complete, the Debugger software will verify that the EPROM matches the image in the Debugger memory. A mismatch will cause an error message, MISMATCH AT XXXX, to be displayed (where XXXX is the byte address). When the flashing cursor appears again at the bottom of the display, verification is complete.

To verify the EPROM, use the Verify command:

V[type][,length][,interleaving factor]

This command compares the EPROM image in the Debugger memory to the EPROM that has just been programmed. A mismatch will cause an error message, MISMATCH AT XXXX, to be displayed (where XXXX is the byte address). This command has the same effect as the automatic verify.

COMMAND PARAMETERS

Type is the type of EPROM being used (2704, 2708, 2716, 2516, 2732, 2532, 2758). It must be an Intel chip or the equivalent. The type defaults to 2708.

Length is the number of bytes in the program or the maximum length of the specific EPROM. The default is the entire length of the EPROM.

Interleaving factor determines how data is programmed into the EPROM from the Debugger memory. It may be a 1 or a 2. When a 1 is used, all addresses will be programmed into the EPROM consecutively. A 2 will cause the contents of alternate memory addresses to be programmed. This enables alternate bytes to be programmed into two EPROMs to form a 16-bit word. The active memory pointer must be positioned to either the first odd or even addresses,

whichever is to be programmed into the EPROM. The default for the interleaving factor is 1.

If a command is entered which changes the value of any one of the parameters, the new value becomes the default for that parameter until either another command is entered using a different value, or the Debugger is reloaded into system memory. For example, if 2732 EPROMs are to be programmed with alternate bytes to form a 16-bit word, the type of EPROM, 2732, and the interleaving factor, 2, need only be specified in the first command (P2732,,2). To program the second through nth EPROMs, position the cursor to the appropriate odd or even address and enter P<RETURN>.

The EPROM Programming commands are also given in the Logic Analyzer/Debugger Reference Card .

SAMPLE EPROM PROGRAMMING PROCEDURE

Place a 2708 EPROM in a 2708 EPROM Programmer module.

Invoke the Debugger by entering the command:

JD<RETURN>

The Logic Analyzer/Debugger must be the only system file starting with a D.

Load the EPROM image from the disk with the Load command:

L{filename}{,offset}

The offset used on the Z80 processor system should be 100, if the EPROM image was linked at 0.

Display the start of the EPROM image loaded in Debugger memory using the D command.

Type C<RETURN> and note if the message EPROM EMPTY is displayed on the screen. If not, replace EPROM.

Enter the Program EPROM command:

P[type][,length][,interleaving factor]

If the length parameter is the length of the EPROM, it may be omitted. The interleaving factor is entered if it is other than 1.

Wait for the cursor to return to the screen.

Enter the command:

V[type][,length][,interleaving factor]

to verify that the EPROM is programmed correctly.

For further information on EPROM programming commands, please refer to **Chapter 5, COMMANDS.**

Chapter 5

COMMANDS

INTRODUCTION

All commands that may be used on the Z80 2300 Advanced Development System are listed and described in this chapter. A list of information messages follows the command section.

FORMATION OF EXPRESSIONS

Many Debugger commands allow expressions to be used as parameters. The elements listed below may be combined to form an expression. Terms may be entered in order but each expression must evaluate to a 16-bit address.

digit [digit][digit][digit]	A hexadecimal constant may be from one to four digits in length.
'character'	An ASCII constant (8-bit character) must be enclosed in single quotes.
+	Addition
-	Subtraction
#[filename:] symbol	Symbolic address. filename is the relocatable object file that was linked to form the absolute object file. filename may be specified in a symbolic address to distinguish between identical address labels.
*	An address term followed by an asterisk evaluates to the contents of the memory location specified (indirect address).

%	An address term followed by a percent sign adds one to the current location. This value is then added to the contents of the current memory location. The percent sign is normally used with relative branch instructions. For example, the command D% is entered. The current location is X'100' and the contents of that location is X'02'. The D% command adds X'100' and X'02' and displays location X'103'.

LIST OF COMMANDS

LOAD PROGRAM [,[offset][,symbol table address[,G]]]

The L command loads the specified absolute object program, modified by an optional offset. If a symbol table is appended to the object file and the symbol-table-address parameter is given, the symbol table will also be loaded. This command sets the program counter to the entry point of the program loaded.

offset specifies a hexadecimal constant or expression to be added to the load address of the object file. If this parameter is omitted, the offset value is set to zero and the program will execute. This parameter is used in the Program EPROM command. (Refer to this chapter, EPROM Programming Commands)

symbol table address may be a hexadecimal constant or an expression specifying the address at which to load the program symbol table. The symbol table address parameter must be separated from filename by two commas if the offset parameter is omitted. If the symbol table address is not specified or is given as zero, the symbol table will not be loaded. G causes only global symbols to be loaded.

STORE DATA IN MEMORY

S constant [,constant]...

Constants are stored in memory one byte at a time, starting at the display cursor address.

constant may be a one- or two-digit hexadecimal number or a single ASCII character enclosed in single quotes. Each character or number in a string must be enclosed in single quotes. If a single digit is used, the high-order four bits are stored as zeros. Constants may be entered until the command line is full. If additional constants must be stored after the command line is filled, the display must be repositioned and another S command issued.

If an attempt is made to store into a write-protected or non-existent portion of memory, the command is ignored. No message will be displayed.

DISPLAY MEMORY

D{expression}
D
D$

The D command displays the specified memory location.

expression is interpreted as an address.

$ returns the previous display to the screen.

In slaved display mode both memory pointers are positioned at the same address, and the D command affects both sides. In split display mode, the D command affects only the active side of the display.

DISPLAY POSITIONING

The up arrow backs up the display eight bytes to a lower memory address.

The down arrow advances the display one instruction higher in memory.

The left arrow backs up the display one byte to a lower memory address.

The right arrow advances the display one byte to a higher memory address.

EXECUTE PROGRAM

E{expression}
E
E$

The E command starts program execution at the specified address with interrupts enabled.

expression is interpreted as an address. If expression is omitted, execution begins at the memory pointer address on the active side of the display.

$ starts execution at the address specified by the program counter.

All subsequent single step and trace commands will start with maskable interrupts enabled.

The Stack Pointer register must contain a RAM address when an E command is entered. If the Stack Pointer register contains ROM, an error message will be displayed.

NO-INTERRUPT EXECUTION COMMAND

N{expression}
N
N$

Execution starts at the address specified by the value of expression. If expression is omitted, execution starts at the memory pointer address of the active side of the display. When execution starts, all maskable interrupts are disabled. This form of the execute command is very useful when debugging interrupt systems and software.

$ starts execution at the address specified in the program counter (register display).

All subsequent single-step and trace commands will start with maskable interrupts enabled.

The Stack pointer register must contain a RAM address when an N command is entered. If the Stack Pointer register contains ROM, an error message will be displayed.

FIND

F[constant][,constant]...

The F command searches for the specified byte or series of bytes.

constant may be a one- or two-digit hexadecimal number or a single ASCII character enclosed in single quotes. Each character or number in a string must be enclosed in single quotes. If a single digit is used, the four high-order bits of that byte are stored as zeros. Constant parameters may be entered until the command line is full.

If constants are not specified, the constants used in the previous F command are used.

The F command begins searching for a match at the memory pointer address on the active side of the display. Unless a match is found, the search continues to the end of memory (FFFF), wraps around to the beginning of memory, and stops at the memory pointer address. When a match is found, the F command positions the memory pointer at the first byte of the match.

SINGLE STEP

The STEP key executes a single user program instruction, allowing register and memory contents to be viewed for each instruction.

STEP executes the instruction at the memory pointer address. If the executed instruction is a branch, jump, or call, the display may show a different area of memory. To restore the former display, enter a D$ command. If the memory pointer is positioned at a breakpoint, D$ has no effect.

SPLIT DISPLAY MODE

TAB

The TAB key instructs the Debugger to work in a split display mode, allowing different areas of memory to be viewed simultaneously.

At the beginning of Debugger operation the memory pointers on each side of the display point to the same address. The first depression of the TAB key selects split display operation. Subsequent depressions of the TAB key move the command cursor to the left or right side which then becomes the active side.

The RESET key followed by a Display Command may be used to return the Debugger to slaved display mode. However, when the RESET key is depressed, all registers except the Stack Pointer will be cleared.

TRACE EXECUTION

T{expression}
T
T$

The T command allows the user to display the last 20 instructions executed prior to a breakpoint.

Trace execution starts at the address specified by expression. If expression is omitted, execution begins at the memory pointer address on the active side of the display.

$ starts trace execution at the address specified by the program counter.

The T command interrupts execution after every instruction to save all register contents. When control of execution is returned to the Debugger (via a breakpoint or depressing <BREAK>), the last 20 traces are shown on the display. Depressing <RETURN> returns the normal Debugger display to the screen. The trace display is saved and may be recalled (see TR command below).

If these are less than 20 instructions executed, only that number of traces (counting from the bottom of the display) are valid. Remaining lines are filled with zeros.

Example

Figure 5-1 shows sample trace execution display. Each line represents the register contents of the Z80 microprocessor after execution of the instruction. The letter register designations are on the top line. The last set of data taken is shown at the bottom of the display.

TRACE RECALL

TR

The TR command recalls the data save in a Trace Execution (T) command. <RETURN> returns the normal Debugger display to the screen.

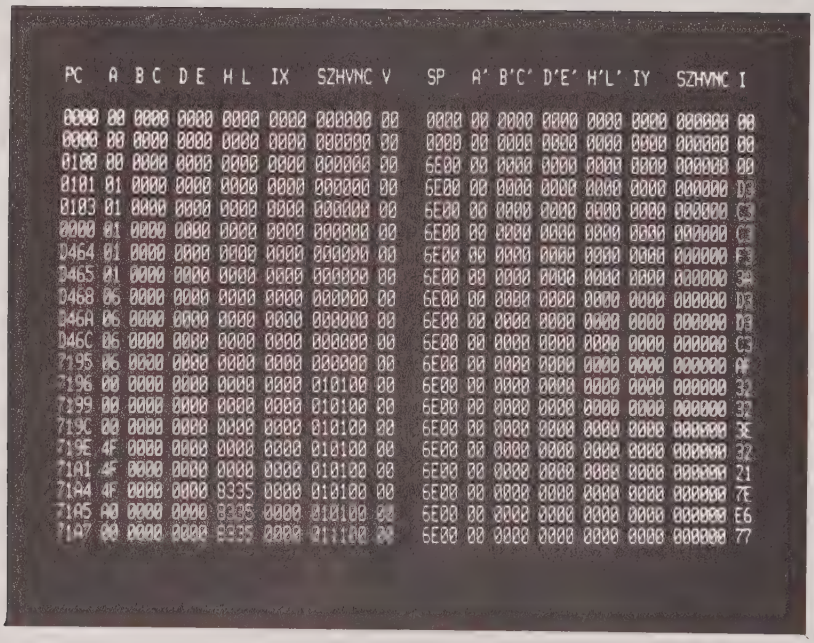


Figure 5-1. Sample Trace Execution Display

WRITE

W{file-spec}{,start addr,end addr}[,start addr,end addr]...[,entry addr]

The W command writes any series of memory locations to a file in absolute object format.

start addr and end addr are hexadecimal constants or expressions.

entry addr specifies a memory location at which execution will begin if file-spec is subsequently loaded and executed. If entry addr is omitted (in other words, if there are an even number of parameters after file-spec), the first start addr is used as the entry point address.

Memory contents are not affected by execution of the W command.

When file-spec is loaded into memory after having been written by the W command, each section is loaded into the location it previously occupied. The program counter is set to the entry point address.

The symbol table may be written and reloaded, but a reloaded symbol table cannot be used for symbolic debugging.

WRITE PROTECT MEMORY

X

The X command sets a flag to prevent the contents of a memory block from being altered.

X switches protection on or off for the 1024-byte (decimal) or 400-byte (hexadecimal) block of memory in which the memory pointer is located. The command is effective only on the active side of the display.

No action is taken and no message is displayed if an attempt is made to alter write-protected memory with a keyboard-entered command or a command file.

STORE DATA IN REGISTER

Z{reg-constant}[,reg-constant]...

The Z command stores keyboard-entered data in the specified register(s).

reg may be any single-letter register designation shown at the top of the display except the program counter. reg may be F to indicate the flag bits (also known as the program status word or condition codes). The flag bits are set by the low-order bits of the specified constant.

If reg is an eight-bit register, constant may be a one- or two-digit hexadecimal number or a single ASCII character enclosed in single quotes. If a single hexadecimal digit is specified, the high-order four bits are stored as zeros.

If reg is a 16-bit register, constant may be a one- to four-digit hexadecimal number, or one or two ASCII characters enclosed in single quotes. If less than four hexadecimal digits or only one ASCII character is specified, the high-order bits are stored as zeros.

EXAMPLE

The command:

ZS=A100,B='H',C=3,F=6

stores X'A100' in the 16-bit Stack pointer register. It also stores the ASCII representation of H (X'48') in the B register. X'03' is stored in the C register. The flag bits X'6' become 0110 (the second and third flags from the right are set high and all others are set low).

BREAKPOINT COMMANDS

SET BREAKPOINT

BS{n}

The BS command sets a breakpoint at the address indicated by the memory pointer on the active side of the display.

n uniquely identifies each breakpoint. n may be an integer between 0 and 4, inclusive. If n=0, a hardware breakpoint is set; if n=1-4, a software breakpoint is set.

Software breakpoints may not be set in write-protected memory.

DISPLAY BREAKPOINT

BD{n}

The BD command displays the location of the specified breakpoint.

n may be an integer between 0 and 4, and must match the breakpoint number specified in a previous BS n command.

RESET BREAKPOINT

BR

The BR command removes the software breakpoint at the address indicated by the active memory pointer and restores the instruction byte formerly at that address. When specified for a hardware breakpoint, the breakpoint is cleared and memory is not affected.

CLEAR BREAKPOINTS

The BC command clears the breakpoint table to allow reloading of the user's program.

The instructions replaced by software breakpoints are cleared from the table but are not restored to memory. Breakpoints remain in memory but cannot be displayed with the BD command. The breakpoints can be found, however, by use of the F command.

INVOKE LOGIC ANALYZER

A

The A command enters the Logic Analyzer from the Debugger.

INVOKE DEBUGGER

D

The D command returns to the Debugger display from the Logic Analyzer.

SELECT AND/OR MODE

A

O

A selects the AND mode of operation in which breakpoints must be satisfied in sequence before data storage is halted.

O selects the OR mode in which data storage is halted when any one of the specified breakpoint conditions have been met.

SET TIMER

T{N / U / B}

The T command sets the timer which measures the length of time between satisfaction of B0 and B1 in the AND condition.

N selects .1 microsecond increments.
U selects 1 microsecond increments.
B selects 1 bus cycle increments.

SET BREAKPOINT IN LOGIC ANALYZER MODE

EXAMPLE

B0,C=10,A=D000,D=C0,RW=W,IO=0,M1=1,DMA=1,E=0101

The B command sets the specified breakpoint and defines the desired breakpoint conditions.

COUNT C allows the user to select the number of times (up to 65,535) the specified condition must occur before the breakpoint can be satisfied.

ADDR A specifies the breakpoint address. This parameter may be a set number consisting of four hexadecimal digits or an expression. A may be set to equal to (=), greater than or equal to (>=) or less than or equal to (<=). Don't care symbols (X) may be specified only with equal to notation (=).

DATA D specifies the value of the data lines. This number is entered as two hexadecimal digits.

RW RW specifies a READ (R) bus cycle, a WRITE (W) bus cycle, or both (X).

IO IO specifies an I/O port (1), a memory address cycle (0), or both (X).

M1 M1 specifies an op-code fetch cycle (1), (0), or both (X).

DMA DMA specifies a DMA cycle (1), a non-DMA cycle (0), or both

EXTR E specifies the value of the four external lines in binary.

When X appears in a hex2 or hex4 number, it represents a Don't care condition for the corresponding four bits. When X is specified in any other parameter, any value will satisfy the breakpoint condition for the parameter.

EXAMPLES

Correct: B0,A>=1025
B0,A=10X4,D=33

Incorrect: B1,A<=10x1 X is not allowed with <=.
B2,A>1040 > is not allowed by itself.

SET TRACE QUALIFIER

EXAMPLE

Q,C=10,A=A100,D=C3,RW=W,IO=1,M1=1,DMA=1,E=110

The Q command selects the traces to be stored in Logic Analyzer memory.

COUNT	C specifies the number of qualified bus cycles (up to 65,535) that will be stored after the Logic Analyzer breakpoints have been met.
ADDR	A specifies the address(es) which may be either a set number consisting of four hexadecimal digits or an expression. This parameter may be set to equal to (=), greater than or equal to(>=), or less than or equal to (<=). Don't care symbols (X) may be specified only with equal to notation (=).
DATA	D specifies the value of the data lines in 2-digit hexadecimal form.
RW	RW specifies a READ (R) bus cycle, a WRITE (W) bus cycle or either (X).
IO	IO specifies an I/O port (1), a memory access cycle (0) or either (X).
M1	M1 specifies a DMA cycle (1), a non-DMA cycle (0) or either (X).
EXTR	E specifies the value of the four external lines in binary.

X represents a Don't care condition for the four bits in EXTR. Individual bits cannot be set in hexadecimal notation; however, in binary form they can be set to X in place of 1 or 0.

EXAMPLES

Correct:	Q,D=37,M1=1	
	Q,C=4,A=0100	
Incorrect:	Q,A>=10XX	X is not allowed with >=
	Q,D<=40	< is not allowed with a D parameter.

SELECT DISPLAY FORMAT

FP
FS[,line name]....

The FP command selects the parallel (cycle-by-cycle) mode which displays 16 traces at a time. Each trace represents the values of the 32 lines during one bus cycle.

The FS command selects the serial (line-by-line) display mode in which the logic states of a selected line may be viewed 64 at a time. Note the following explanation of Display Positioning for use of the arrow keys and the P command.

EXAMPLE

```
FS,A13,D7,IO,
```

PARAMETERS

FS[, line name] selects any of the following lines for viewing:

A15	most significant bit of A	D07	most significant bit of D
A14		D06	
A13		D05	
A12		D04	
A11		D03	
A10		D02	
A09		D01	
A08		D00	least significant bit of D
A07		IO	
A06		RW	
A05		DMA	
A04		M1	
A03		E0	
A02		E1	
A01		E2	
A00	least significant bit of A	E3	

EMULATOR MODE COMMAND

M [mode] [mode] ...

mode may be any combination of the following:

C Enables the user system clock line input(s). This mode causes the Emulator to switch from the ADS internal clock to the target system clock.

D Allows the target system to generate DMA requests to suspend ADS microprocessor operation. This allows user DMA logic to gain control of the bus and generate transactions to target system memory. (Note: User DMA logic cannot access ROM/PROM simulator memory). For the Z80 processor the DMA request is caused by a low level (0.4 VDC) on the BUSRQ- line and the DMA grant signal is indicated by a low level on the BUSAK- line.

M Enables the memory mapping function. (see the USERMAP command).

I Allows the target system to control the following input lines: RESET-, INT-, WAIT-, and NMI-. The normal (inactive) conditions for these lines are:

```
RESET- = high ( 2.4 VDC)
INT-   = high
WAIT-  = high
NMI-   = high
```

Note: A low on the WAIT- line will completely halt the microprocessor.

The I mode parameter also enables the emulator to direct I/O requests to the target system. The user must be aware that output port X'F6' is reserved for switching emulator control functions and is not available to the user.

E The E mode is a combination of C, I, M, and D.

R The R mode will disable the CRT and Memory refresh during user program execution. It is not possible to use this mode parameter without the static RAM option, since disrupting the refresh will cause both the user program and the Debugger to be lost if they are stored in dynamic RAM.

If the M command is entered without any mode parameters, the system will return to full host mode and all inputs, clocks, memory and I/O requests will be directed to the host system.

The user should be aware of the exact timing of the various mode switches. The C and D mode parameters cause an immediate switch of the affected control lines as soon as the carriage return is typed following MC, MD, MCD etc. The R mode parameter does not switch off the refresh function until the user program is executed. The refresh function is switched back on as soon as control is returned to the Debugger after a break from the user program. The I mode parameter will enable the interrupt, reset, ready control lines and I/O requests when the user progam is executed, and disable them when the control is returned to the Debugger. The M mode parameter will enable the memory mapping function whenever a memory access is made (for example: to display memory, alter memory, execute a user program etc.).

USER MEMORY MAP COMMAND

U[addr], [mask]

The user must first input an MM or an ME command to enable the User Memory Map. Refer to Emulator Mode command for details. The U command transfers the memory space specified by mask starting at addr into the target system. When a memory location is accessed that has been mapped into the user system, the memory request and read or write will be directed to the target system instead of the host ROM/PROM simulator. All addresses may be mapped into the user system except those privileged locations reserved by the Debugger communication block (DEBG256). (See section, **Considerations Prior to Using the Z80 In-Circuit Emulator, Chapter 2.**)

Each user map begins on a 2K boundary. Each bit in the mask switches 256 bytes into the user system. The bits are individually "OR'ed" to form the mask. Thus U1800, 0F would switch the memory space from X'1800' to X'1BFF' into the user system. If addr is not on a 2K boundary then the message "NOT ON 2K BOUNDARY" will be displayed. If the addr parameter is omitted, the map will begin at the present cursor location.

2K Boundaries

0000	4000	8000	C000
0800	4800	8800	C800
1000	5000	9000	D000
1800	5800	9800	D800
2000	6000	A000	E000
2800	6800	A800	E800
3000	7000	B000	F000
3800	7800	B800	F800

EPROM PROGRAMMING COMMANDS

CHECK FOR EMPTY

C[type][,length]

Before programming, an EPROM must be completely erased. The C command verifies that the EPROM contains no information.

PROGRAM EPROM

P[type][,length[,interleaving factor]]

The P command programs the EPROM with data beginning at the current cursor address.

INPUT DATA FROM EPROM

I[type][,length[,interleaving factor]]

The I command reads data from the EPROM into memory beginning at the current cursor location.

VERIFY DATA IN EPROM

V[type][,length[,interleaving factor]]

Beginning at the current cursor location and at location 0 in the EPROM, THE V command compares data in the EPROM to data in memory.

INFORMATION/ERROR MESSAGES

IN USE

An attempt was made to set a breakpoint using a number already in use, or an attempt was made to set a software breakpoint in write-protected memory. The breakpoint is set in the breakpoint table but not in memory. Thus, it appears from a memory display that the breakpoint command was not executed. The second attempt to set the same breakpoint causes this message to be displayed.

NAME ERR

A file-spec was entered with incorrect syntax.

NOT FOUND

A Find command did not find the specified byte in memory.

NOT IN USE

1. An attempt was made to display or reset a numbered software breakpoint not in use.

2. An attempt was made to reset a breakpoint and the memory pointer on the active side of the display was not positioned at a breakpointed instruction.

NOT OBJECT FILE

The file specified in a Load or Write command does not have the "O" attribute designating it as an absolute object file.

SP NOT IN RAM

An attempt was made to execute a program when the Stack Pointer was not in random access memory (RAM).

SYMTAB LENGTH = XXXX

This message is displayed when program loading with a symbol table address parameter has been completed. XXXX indicates the length of the symbol table. If the symbol table was not loaded, the length is given as zero.

GLOBAL SYMTAB LENGTH = XXXX

Same as above, but for global symbols.

SYMTAB LENGTH =XXXX PARTLY LOADED

There is insufficient memory space for the symbol table. Reload at a lower address.

GLOBAL SYMTAB LENGTH = XXXX PARTLY LOADED

Same as above but for global symbols.

SYNTAX

A syntax error was detected in a command.

UNDEFINED SYMBOL

An expression in a command parameter contains a symbol which is not in the symbol table. Retype the parameter.

EPROM IS EMPTY

The EPROM tested by a C command is empty.

MISMATCH AT XXXX

The EPROM does not match memory (P and V commands) at location X'XXXX'.

NOT EMPTY AT XXXX

The EPROM tested by a C command is not empty at location X'XXXX'.

GENRAD/DSD SERVICE LOCATIONS - USA & CANADA

CO	300 Baker Avenue Concord, MA 01742 617/369-4400 or 617/646-7400 TWX: 710-347-1051	Conn, Maine, Mass., N. H., R. I., Vt.
CHO	1083 East State Parkway Schaumburg, IL 60195 312/843-5580 TWX: 910-291-1209	Ill, Iowa, Minn, Ohio, Mo., N.D., S.D., Ky., Ind., Mich., W. Pa., Wis., Neb.
DEN	13132 St. Paul Drive Thornton, CO 80241 303/457-9147 (ans.serv.)	Colo., Mont., Wyoming, N.M., Utah
DSD	5730 Buckingham Parkway Culver City, CA 90230 (Factory) 213/641-7200; TWX: 910-328-7202	Factory Field Support
DTX	1121 Rockingham, Suite 100, Richardson, TX 75080 214/234-3357; TWX: 910-867-4771	Ark., Kan., La., Tex., Okla.
FLO	3751 Maguire Blvd Suite 170, Orlando, FL 32803 305/894-4303; TWX: 810-850-0270	Ala., Ga., Fla., N.C., S.C., Tenn, Miss.
LAO	17631 Armstrong Avenue P.O. Box 19500,Irvine, CA 92714 714/540-9830; TWX: 910-595-1762	So. Calif., Ariz., So. Nev.
NYO	380 Midland Avenue Saddlebrook, NJ 07662 201/791-8999 (NJ); 212/964-2722 (NY) TWX: 710-988-2205	Del., E. Pa., N.J., N.Y.
SFO	2855 Bowers Avenue, Santa Clara, CA 95051 408/727-4400; TWX: 910-338-0291	No. Calif., Idaho, No. Nev., Wash., Ore.
WO	1701 Research Blvd, Rockville, MD 20850 301/424-6224; TWX: 710-828-9783	Md., Va., W. Va., D.C.
GRC	307 Evans Avenue Toronto, Ontario, Canada M8Z 1K2 416/252-3395; TELEX: 06-967624	All Canada

GENRAD/DSD SERVICE LOCATIONS - EUROPE

GRU NORTHERN REGION GENRAD, LTD.
Norreys Drive, Maidenhead
Berkshire SL6 48P, England
Tele: (628) 39181 Tele: 851-848321

GRF SOUTHERN REGION GENRAD, INC.
96 Rue Orfila
75020 Paris, France
Tele: (01)7970739 Telex: 842-220991

GRG EASTERN REGION GENRAD, GmbH
Trausnitzstrasse 8
D-8 Munich 80, West Germany
Tele: (089) 402801 Telex: 841-529917

GRS EASTERN REGION GENRAD, (SCHWEIZ) AG
Helenastrasse 3
8034 Zurich, Switzerland
Tele: (01) 552420 Telex: 845-53638

GRB EASTERN REGION GENRAD/BENELUX B.V.I.O.
P.O. Box 210
5658 Aebest, Netherlands
Tele: (04) 9984240 Telex: 59535 GRBEN NL

GRI EASTERN REGION GENRAD S.P.A.
Via Lampedusa 13
20141 Milano, Italy
Tele: (02) 8466541 Telex: 843-320373

GenRad
Development Systems Division

Technical Publications Remarks Form

Please use this form to submit your suggestions for revisions, corrections, or additions to this publication. Your comments will be promptly investigated by appropriate technical personnel, and action will be taken as required. If your answer to any of the questions is 'NO' or requires qualification, please include any additional comments on a separate sheet and enclose it inside this pre-addressed form.

TITLE: _____

ORDER NO. _____

DATED _____

	YES	NO
DOES THIS DOCUMENT MEET YOUR NEEDS?	_____	_____
IS THE MATERIAL CONTAINED IN THIS DOCUMENT:		
ACCURATE?	_____	_____
EASY TO READ AND UNDERSTAND?	_____	_____
ORGANIZED FOR CONVENIENT REFERENCING?	_____	_____
WELL ILLUSTRATED WITH USEFUL EXAMPLES?	_____	_____
COMPLETE?	_____	_____

ERRORS IN THE PUBLICATION

SUGGESTIONS FOR IMPROVEMENT

(Please Print)

FROM: NAME _____ DATE _____

ORGANIZATION _____

ADDRESS _____

TITLE _____

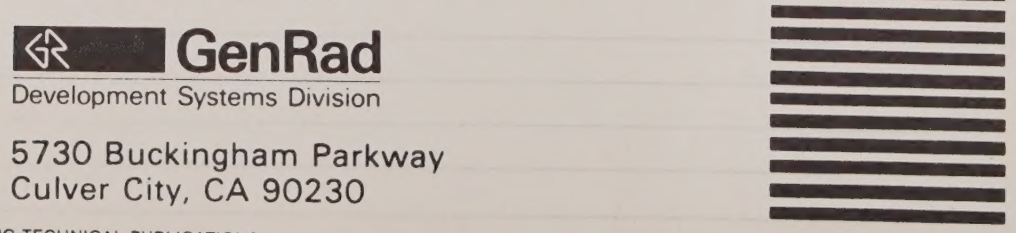
CUT ALONG DOTTED LINE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 608 CULVER CITY, CA

POSTAGE WILL BE PAID BY ADDRESSEE



GenRad

Development Systems Division

5730 Buckingham Parkway
Culver City, CA 90230

ATTN: MARKETING TECHNICAL PUBLICATIONS